1发展过程

2018年11月21日 14:30

-
- 操作系统的发展过程

一. 无操作系统的计算机系统

1. 人工操作方式

1) 由程序员将事先已穿孔(对应于程序和数据)的纸带(或卡片)装入纸带输入机(或卡片输入机)，再启动它们将程序和数据输入计算机，然后启动计算机运行。当程序运行完毕并取走计算结果之后，才让下一个用户上机
2) 有以下两方面的缺点:
   (1) 用户独占全机。此时，计算机及其全部资源只能由上机用户独占
   (2) CPU 等待人工操作。当用户进行装带(卡)、卸带(卡)等操作时，CPU 及内存等资源是空闲的

2. 脱机输入/输出方式(Off-Line I/O)

1) 事先将装有用户程序和数据的纸带/卡片装入纸带输入机/卡片机，在外围机控制下，把纸带/卡片上的数据/程序输入到磁带。当 CPU 需要程序/数据时，再从磁带上高速地调入内存。同理输出

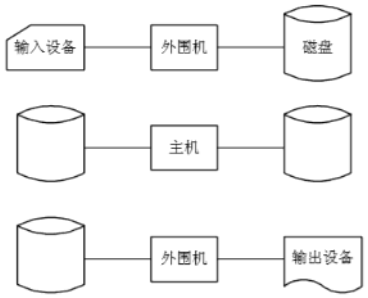


图 1-3 脱机 I/O 示意图

2) 优点如下:
   (1) 减少了CPU的空闲时间，缓和了人机矛盾: 装带(卡)、 卸带(卡)以及将数据从低速 I/O 设备送到高速磁带(或盘)上，都是在脱机情况下进行的，不占用主机时间，有效减少了 CPU 空闲时间
   (2) 提高了 I/O 速度，缓和了 CPU 和 I/O 设备速度不匹配的矛盾，进一步减少 CPU 空闲时间: 当 CPU 在运行中需要数据时，是直接从高速的磁带或磁盘上将数据调入内存的，不再是从低速 I/O 设备上输入

二. 单道批处理系统(Simple Batch Processing System)

1. 单道批处理系统的处理过程

1) 晶体管替代真空管制作出第二代计算机。使计算机的体积大大减小，功耗显著降低，同时可靠性也得到大幅度提高，使计算机已具有推广应用的价值，但计算机系统仍非常昂贵
2) 为减少空闲时间，通常把一批作业以脱机方式输入到磁带上，并在系统中配上监督程序(Monitor)
3) 处理过程是: 先由监督程序将磁带上的第一个作业装入内存，并把运行控制权交给该作业。当该作业处理完成时，又把控制权交还给监督程序，再由监督程序把磁带(盘)上的第二个作业调入内存。计算机系统就这样自动地一个作业一个作业地进行处理，直至磁带(盘)上的所有作业全部完成

4)

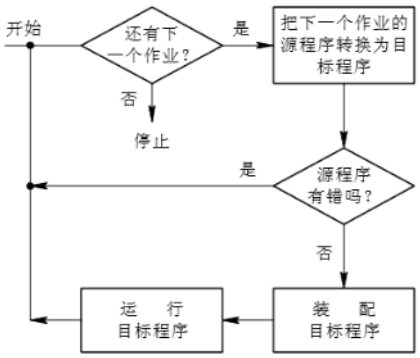


图 1-4 单道批处理系统的处理流程

2. 单道批处理系统的特征

1) 它只能算作是 OS 的前身而并非是现在人们所理解的 OS。尽管如此，该系统比起人工操作方式的系统已有很大进步。该系统的主要特征如下：

(1) 自动性：在顺利情况下，磁带上的一批作业能自动地逐个地依次运行，无需人工干预

(2) 顺序性：磁带上的各道作业是顺序地进入内存，各道作业的完成顺序与它们进入内存的顺序，在正常情况下应完全相同，亦即先调入内存的作业先完成

(3) 单道性：内存中仅有一道程序运行，即监督程序每次从磁带上只调入一道程序进入内存运行，当该程序完成或发生异常情况时，才换入其后继程序进入内存运行

三. 多道批处理系统(Multiprogrammed Batch Processing System)

1) 20世纪60年代中期，IBM360的OS/360是第一个多道批处理系统

1. 多道程序设计的基本概念

1) 小规模集成电路生产出了第三代计算机，在体积、功耗、速度和可靠性上，都有了显著的改善

2) 用户提交的作业都先存放在外存上并排成一个队列，称为“后备队列”；由作业调度程序按一定的算法选择若干作业调入内存，使它们共享 CPU 和系统中的各种资源。引入多道程序设计技术可带来以下好处：

(1) 提高 CPU 的利用率：由于同时在内存中装有若干道程序，并使它们交替地运行，这样，当正在运行的程序因 I/O 而暂停执行时，系统可调度另一道程序运行，从而保持了 CPU 处于忙碌状态

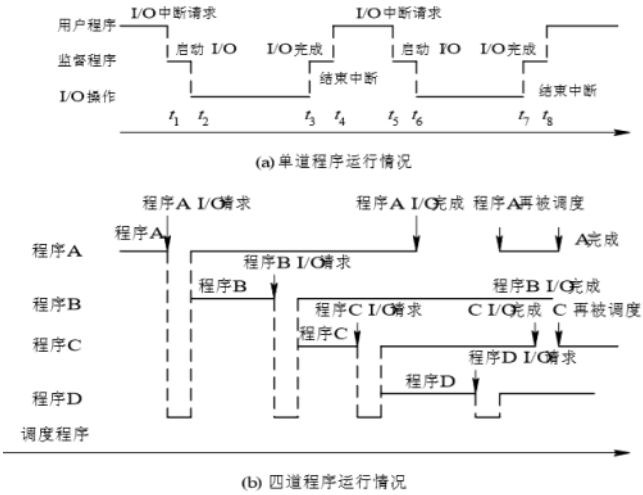


图 1-5 单道和多道程序运行情况

(2) 提高内存和 I/O 设备利用率：允许多道程序装入内存，并发执行

(3) 增加系统吞吐量：在保持 CPU、I/O 设备不断忙碌的同时，也必然会大幅提高系统的吞吐量，从而降低作业加工所需的费用

2. 多道批处理系统的优缺点

(1) 至今它仍是三大基本操作系统类型之一。在大多数大、中、小型机中都配置了它，说明它

具有其它类型 OS 所不具有的优点。多道批处理系统的主要优缺点如下：

1) 资源利用率高：由于在内存中驻留了多道程序，它们共享资源，可保持资源处于忙碌状态，从而使各种资源得以充分利用
2) 系统吞吐量大：
   (1) 系统吞吐量是指系统在单位时间内所完成的总工作量
   (2) 能提高系统吞吐量的主要原因可归结为：
      i. 第一，CPU 和其它资源保持“忙碌”状态；
      ii. 第二，仅当完成或运行不下去时才进行切换，系统开销小
3) 平均周转时间长：作业的周转时间是指从作业进入系统开始，直至其完成并退出系统为止所经历的时间。在批处理系统中，由于作业要排队，依次进行处理，因而作业的周转时间较长，通常需几个小时，甚至几天
4) 无交互能力：用户一旦把作业提交给系统后，直至作业完成，都不能与自己的作业进行交互，这对修改和调试程序是极不方便的

3. 需要解决的问题
   1) 处理机管理问题：在多道程序之间，如何分配被它们共享的处理机，使 CPU 既能满足各程序运行的需要，又能提高处理机利用率，以及处理机分配给程序后， 应在何时收回等问题
   2) 内存管理问题：如何为每道程序分配必要的内存空间，使它们“各得其所”且不致因相互重叠而丢失信息，以及应如何防止因某道程序出现异常情况而破坏其它程序等问题
   3) I/O 设备管理问题：如何分配供多道程序所共享的各种I/O 设备，如何做到既方便用户对设备的使用，又能提高设备的利用率
   4) 文件管理问题：大量的程序和数据(以文件形式存在)，应如何组织，才能使它们既便于用户使用，又能保证数据的安全性和一致性
   5) 作业管理问题：对于系统中的各种应用程序，其中有的以计算为主；有的以 I/O 为主；又有些作业既重要又紧迫；而有的作业则要求系统能及时响应，这时应如何组织这些作业
4. 据此，我们可定义：**操作系统是一组控制和管理计算机硬件和软件资源，合理地对各类作业进行调度，以及方便用户使用的程序的集合**

四. 分时系统(Time Sharing System)

1) 第一台真正的分时操作系统(CTSS，Compatable Time Sharing System)是麻省理工学院开发的。之后，麻省理工学院又和贝尔实验室、通用电气公司联合开发出多用户多任务操作系统——MULTICS，该机器能支持数百用户。参加研制的贝尔实验室的 Ken Thempson，在 PDP-7 小型机上开发出一个简化版 MULTICS，是当今广为流行的 UNIX 操作系统的前身

1. 分时系统的产生
   1) 用户的需求具体表现在以下几个方面：
      (1) 人-机交互：由于新编程序难免有些错误或不当之处需要修改，因而希望能像早期使用计算机时一样对它进行直接控制，并能以边运行边修改的方式，对程序中的错误进行修改
      (2) 共享主机：在 20 世纪 60 年代计算机非常昂贵，只能多个用户共享一台计算机，但用户在使用机器时应能够像自己独占计算机一样，不仅可以随时与计算机交互，而且应感觉不到其他用户也在使用该计算机
      (3) 便于用户上机：在多道批处理系统中，用户上机前必须把作业邮寄或亲自送到机房。用户希望能通过自己的终端直接将作业传到机器上进行处理，并能对自己的作业进行控制
2. 分时系统实现中的关键问题
   1) 及时接收：只需在系统中配置一个多路卡，每个终端配一个缓冲区
      (1) 例如，要在主机上连接 8 个终端时，须配置一个 8 用户的多路卡
      (2) 多路卡使主机能同时接收各用户从终端上输入的数据
      (3) 缓冲区用来暂存用户键入的命令(或数据)
   2) 及时处理：

(1) 各个用户的作业都必须在内存中，且应能频繁获得处理机而运行
(2) 通常大多数作业都还驻留在外存上，即使是已调入内存的作业，也经常要经过较长时间的等待后方能运行，因而使用户键入的命令很难及时作用到自己的作业上

3. 分时系统的特征
   1) 多路性/同时性：允许在一台主机上同时联接多台联机终端，系统按分时原则为每个用户服务。宏观上，是多个用户同时工作，共享系统资源；微观上，每个用户作业轮流运行一个时间片。提高了资源利用率，降低了使用费用，促进了计算机更广泛的应用
   2) 独立性：每个用户各占一个终端，彼此独立操作，互不干扰。用户所感觉到的，就像是他一人独占主机
   3) 及时性：用户的请求能在很短的时间内获得响应。此时间间隔是以人们所能接受的等待时间来确定的，通常仅为 1～3 秒钟
   4) 交互性：用户可通过终端与系统进行广泛的人机对话。其广泛性表现在：用户可以请求系统提供多方面的服务，如文件编辑、数据处理和资源共享等
4. 在批处理兼分时的系统中，往往由分时系统控制的称为前台作业，而由批处理系统控制的称为后台作业

五. 实时系统

1) 实时/及时计算：不仅要逻辑结果正确，还要在规定时间内算出

1. 实时系统的类型：
   1) 实时控制：能实时采集现场数据，并对所采集的数据进行及时处理
      (1) 通常把用于进行实时控制的系统称为实时系统
   2) 实时信息处理：用于对信息进行实时处理的系统
      (1) 该系统由一台或多台主机通过通信线路连接到成百上千个远程终端上，计算机接收从远程终端上发来的服务请求，根据用户提出的请求对信息进行检索和处理，并在很短的时间内为用户做出正确的响应
      (2) 典型的实时信息处理系统有早期的飞机或火车的订票系统、 情报检索系统等
   3) 多媒体系统：播放音视频的系统
   4) 嵌入式系统：将集成电路芯片嵌入到各种仪器和设备中，构成所谓的智能仪器和设备。在这些设备中配置的、实时控制的系统
2. 实时任务：若干个实时任务与某个(些)外部设备相关，能反应或控制相应的外部设备，因而带有某种程度的紧迫性
   1) 按任务执行时是否呈现周期性来划分
      (1) 周期性实时任务。外部设备周期性地发出激励信号给计算机，要求它按指定周期循环执行，以便周期性地控制某外部设备
      (2) 非周期性实时任务。外部设备所发出的激励信号并无明显的周期性，但都必须联系着一个截止时间(Deadline)。它又可分为开始截止时间(在某时间以前必须开始执行)和完成截止时间(在某时间以前必须完成)
   2) 根据对截止时间的要求来划分
      (1) 硬实时任务(Hard real-time Task)。系统必须满足任务对截止时间的要求，否则可能出现难以预测的结果
      (2) 软实时任务(Soft real-time Task)。它也联系着一个截止时间，但并不严格，若偶尔错过了任务的截止时间，对系统产生的影响也不会太大
3. 实时系统vs分时系统

	实时控制	实时信息	分时系统
多路性	周期性对多路现场信息进行采	按分时原则为多个终端	与用户情况有关，

	集，以及对多个对象或多个执行机构进行控制	用户服务	时多时少
独立性	对信息的采集和对对象的控制也都是彼此互不干扰	各终端在向实时系统提出服务请求时，是彼此独立地操作，互不干扰	每个用户各占一个终端，彼此独立操作，互不干扰
及时性	以控制对象所要求的开始/完成截止时间来确定，一般为秒级到毫秒级，甚至低于100微秒	以人所能接受的等待时间来确定的	以人所能接受的等待时间来确定的
交互性	能向终端用户提供数据处理和资源共享等服务	仅限于访问系统中某些特定的专用服务程序	用户可通过终端与系统进行广泛的人机对话
可靠性	要求系统具有高度可靠性，任何差错都可能带来巨大的经济损失，甚至是无法预料的灾难性后果。接右	接左，往往采取了多级容错措施来保障系统的安全性及数据的安全性	也要求系统可靠

六. 微机操作系统的发展

1) 随VLSI 和计算机体系结构的发展，以及应用需求的不断扩大，操作系统仍在继续发展。配置在微型机上的操作系统称为微机操作系统。最早的微机操作系统是配置在 8 位微机上的 CP/M。后来16 位，32 位和 64 位微机操作系统也应运而生。微机操作系统可按微机的字长分，也可按运行方式分为以下几类：

1. 单用户单任务操作系统：只允许一个用户上机，且只允许用户程序作为一个任务运行。最简单的微机操作系统，主要配置在 8 位和 16 位微机。最有代表性的是 CP/M 和 MS-DOS

1) CP/M 1974 年第一代通用 8 位微处理机芯片 Intel 8080 出现后的第二年，Digital Research 公司 就开发出带有软盘系统的 8 位微机操作系统。1977 年 Digital Research 公司对 CP/M 进行了 重写，使其可配置在以 Intel 8080、8085、Z80 等 8 位芯片为基础的多种微机上。1979 年又推出带有硬盘管理功能的 CP/M 2.2 版本。由于 CP/M 具有较好的体系结构，可适应性强， 且具有可移植性以及易学易用等优点，使之在 8 位微机中占据了统治地位

2) MS-DOS 1981 年 IBM 公司首次推出了 IBM-PC 个人计算机(16 位微机)，在微机中采用了微软公司开发的 MS-DOS(Disk Operating System)操作系统，该操作系统在 CP/M 的基础上进行了较 大的扩充，使其在功能上有很大的增强。1983 年 IBM 推出 PC/AT(配有 Intel 80286 芯片)， 相应地，微软又开发出 MS-DOS 2.0 版本，不仅能支持硬盘设备，还采用了树形目录结构的文件系统。1987 年又宣布了 MS-DOS 3.3 版本。从 MS-DOS 1.0 到 3.3 为止的 DOS 版本 都属于单用户单任务操作系统，内存被限制在 640 KB。从 1989 年到 1993 年又先后推出了 多个 MS-DOS 版本，它们都可以配置在 Intel 80386、80486 等 32 位微机上。从 20 世纪 80 年代到 90 年代初，由于 MS-DOS 性能优越而受到当时用户的广泛欢迎，成为事实上的 16 位单用户单任务操作系统标准

2. 单用户多任务操作系统：只允许一个用户上机，但允许用户把程序分为若干个任务，使它们并发执行，有效地改善了系统的性能。目前在 32 位微机上配置的操作系统基本上都是单用户多任务操作系统，最有代表性的是由微软公司推出的 Windows。 1985 年和 1987 年微软公司先后推出了 Windows 1.0 和 Windows 2.0 ，由于当时的硬件平台还只是 16 位微机，对 1.0 和 2.0 版本不能很好的支持。1990 年微软公司又发布了 Windows 3.0 版本，随后又宣布了 Windows 3.1 版本，它们主要针对 386 和 486 等 32 位微机开发的，较之以前的操作系统有着重大的改进，引入了友善的图形用户界面，支持多任务和扩展内存的功能，使计算机更好使用，从而成为 386 和 486 等微机的主流操作系统。 1995 年微软公司推出了 Windows 95，它较之以前的 Windows 3.1 有许多重大改进，采用了全 32 位的处理技术，并兼容以前的 16 位应用程序，还集成了支持 Internet 的网络功能。1998 年微软公司又推出了 Windows 95 的改进版 Windows 98，是最后一个兼容以前的 16 位应用程序的 Windows，其最主要的改进是把微软公司自己开发的 Internet 浏览器整合到系统中，大大方便了用户上网浏览，另一个特点是增加了对多媒体的支持。2001 年微软又发布了 32 位版本和 64 位版本的 Windows XP，同时提供了家用和商业工作

站两种版本，是当前使用最广泛的个人操作系统。在开发上述 Windows 操作系统的同时，微软公司又开始开发网络操作系统 Windows NT，是针对网络开发的操作系统，融入了许多面向网络的功能

3. 多用户多任务操作系统：允许多个用户通过各自的终端使用同一台机器，共享主机系统中的各种资源，而每个用户程序又可进一步分为几个任务，使它们能并发执行，从而进一步提高资源利用率和系统吞吐量。在大、中和小型机中所配置的大多是多用户多任务操作系统，而在 32 位微机上也有不少是配置的多用户多任务操作系统，其中最有代表性的是 UNIX OS。UNIX OS 是美国电报电话公司的 Bell 实验室在 1969～1970 年期间开发的，1979 年推 出来的 UNIX V.7 已被广泛应用于多种中、小型机上。随着微机性能的提高，人们又将 UNIX 移植到微机上。在 1980 年前后，将 UNIX 第 7 版本移植到 Motorola 公司的 MC 680xx 微机上，后来又将 UNIX V7.0 版本进行简化后移植到 Intel 8080 上，把它称为 Xenix。现在最有影响的两个能运行在微机上的 UNIX 操作系统的变型是 Solaris OS 和 Linux OS。
    1) Solaris OS：SUN 公司于 1982 年推出的 SUN OS 1.0 是一个运行在 Motorola 680x0 平台上的 UNIX OS。在 1988 年宣布的 SUN OS 4.0 把运行平台从早期的 Motorola 680x0 平台迁移到 SPARC 平台，并开始支持 Intel 公司的 Intel 80x86；1992 年 SUN 发布了 Solaris 2.0。 从 1998 年开始，Sun 公司推出 64 位操作系统 Solaris 2.7 和 2.8，这几款操作系统在网络特性、互操作性、兼容性以及易于配置和管理方面均有很大的提高
    2) Linux OS：Linux 是 UNIX 的一个重要变种，最初由芬兰学生 Linus Torvalds 针对 Intel 80386 开发的。1991 年在 Internet 网上发布第一个版本，由于源代码公开，很多人通过 Internet 与之合作，使 Linux 的性能迅速提高，其应用范围也日益扩大。相应地，源代码也急剧膨胀，此时它已是具有全面功能的 UNIX 系统，大量在 UNIX 上运行的软件(包括 1000 多种实用工具软件和大量的网络软件)被移植到 Linux 上，且可以在主要的微机上运行，如 Intel 80x86 Pentium
        i.
        ii.
        iii.
        iv.
        v.
        vi. --------我是底线---------

1目标、作用、特性

2018年9月19日 13:21

-
- 操作系统的目标和作用

1. 操作系统是计算机硬件上的第一层软件，是首次扩充，是**系统软件**

一. 操作系统的目标

1. 有效性
   1) 提高系统资源利用率：使CPU和I/O设备由于能保持忙碌状态而得到有效的利用，且可使内存和外存中存放的数据因有序而节省了存储空间
   2) 提高系统的吞吐量：通过合理地组织计算机的工作流程，而进一步改善资源的利用率，加速程序的运行，缩短程序的运行周期，从而提高单位时间内完成的作业数
2. 方便性
   1) 通过OS所提供的各种命令来使用计算机系统。比如，用编译命令可方便地把用户用高级语言书写的程序翻译成机器代码，大大地方便了用户，从而使计算机变得易学易用
3. 可扩充性
   1) 随着VLSI技术和计算机技术的迅速发展，计算机硬件和体系结构也得到迅速发展，相应地，它们也对OS提出了更高的功能和性能要求
   2) 多处理机系统、计算机网络，特别是Internet的发展，又对OS提出了一系列更新的要求
4. 开放性
   1) 为使来自不同厂家的计算机和设备能通过网络加以集成化，并能正确、有效地协同工作，实现应用的可移植性和互操作性，要求操作系统必须提供统一的开放环境，进而要求OS具有开放性
   2) 凡遵循开放系统互连(OSI)国际标准所开发的硬件和软件，均能彼此兼容，可方便地实现互连
   3) 开放性已成为20世纪90年代以后计算机技术的一个核心问题，也是新推出的系统或软件能否被广泛应用的至关重要的因素

二. 操作系统的作用

1. 用户与计算机硬件系统之间的接口

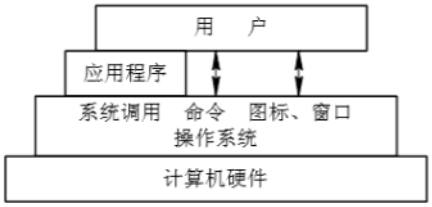


图 1-1 OS 作为接口的示意图

   1) 用户可通过以下三种方式使用计算机
      (1) 命令方式：由OS提供了一组联机命令接口，以允许用户通过键盘输入有关命令来取得操作系统的服务，并控制用户程序的运行
      (2) 系统调用方式：OS提供了一组系统调用，用户可在自己的应用程序中通过相应的系统调用，来实现与操作系统的通信，并取得它的服务
      (3) 图形、窗口方式：通过屏幕上的窗口和图标实现与操作系统的通信，并取得服务
2. 计算机系统资源的管理者
   1) 资源：处理器、存储器、I/O设备以及信息(数据和程序)
   2) 是当今世界上广为流行的一个关于OS作用的观点
3. 实现计算机资源的抽象

1)

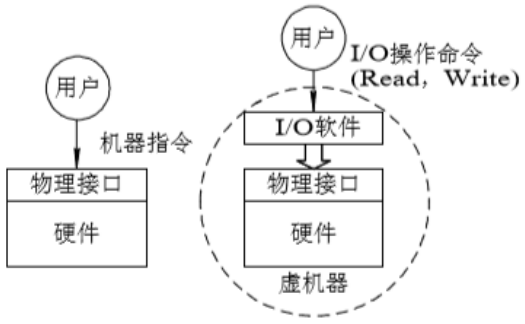


图 1-2　I/O 软件隐藏了 I/O 操作实现的细节

2) 通常把覆盖了上述软件的机器称为扩充机器或虚机器。它向用户(进程)提供了一个对硬件操作的抽象模型，用户可利用抽象模型提供的接口使用计算机，而无需了解物理接口实现的细节，从而使用户更容易地使用计算机硬件资源
3) 为了方便用户使用文件系统，人们又在第一层软件上再覆盖上一层用于文件的管理软件，同样由它来实现对文件操作的细节，并向上提供一组对文件进行存取操作的命令，用户可利用这组命令进行文件的存取。该层软件实现了对硬件资源操作的第二个层次的抽象
4) 而当人们又在文件管理软件上再覆盖一层面向用户的窗口软件后，用户便可在窗口环境下方便地使用计算机，形成一台功能更强的虚机器
5) 对一个硬件在底层进行抽象后，在高层还可再次对该资源进行抽象，成为更高层的抽象模型。随着抽象层次的提高，抽象接口所提供的功能就越来越强，用户使用起来也更加方便

三. 推进发展的主要动力

1. 提高计算机资源利用率

1) 计算机发展的初期，计算机系统特别昂贵，人们必须千方百计地提高计算机系统中各种资源的利用率，这就是 OS 最初发展的推动力
2) 形成了能自动地对一批作业进行处理的多道批处理系统

2. 方便用户

1) 利用率问题基本解决后，用户在上机、调试程序时的不方便性便又成为主要矛盾
2) 便形成了允许进行人机交互的分时系统，或称为多用户系统

3. 器件更新换代

1) 微电子技术的迅猛发展，推动着计算机器件，特别是微机芯片的不断更新，使得计算机的性能迅速提高，规模急剧扩大，从而推动了 OS 的功能和性能也迅速增强和提高
2) 外部设备也在迅速发展。例如，磁盘价格的不断降低且小型化，很快在中、小型机以及微型机上也无一例外地配置了磁盘系统。 现在的微机操作系统(如 Windows XP)能支持种类非常多的外部设备，除了传统的外设外， 还可以支持光盘、移动硬盘、闪存盘、扫描仪等

4. 计算机体系结构的发展

1) 如，计算机由单处理机系统发展为多处理机系统，操作系统也由单处理机 OS 发展为多处理机 OS
2) 又如，当出现了计算机网络后，配置在计算机网络上的网络操作系统也就应运而生，它不仅能有效地管理好网络中的共享资源，而且还向用户提供了许多网络服务

5. 新应用需求的提出

1) 如：工业控制：实时系统；娱乐：多媒体功能；信息保存：安全性

-
- 操作系统的基本特性

并发、共享、虚拟、异步

一. 并发性Concurrence

1. **并发**性：两个或多个事件在**同一时间间隔内**发生；

1) 并行性：两个或多个事件在同一时刻发生
2) 在多道程序环境下，并发性是指在一段时间内宏观上有多个程序在同时运行

3) 在单处理机系统中，每一时刻却仅能有一道程序执行，故微观上这些程序只能是分时地交替执行。多处理机系统，则可并发执行程序，将其分配到多个处理机上，实现并行执行，即利用每个处理机来处理一个可并发执行的程序，这样，多个程序便可同时执行

2. 引入进程
   1) 通常的程序是静态实体(Passive Entity)，在多道程序系统中，不能独立运行，更不能和其它程序并发执行。引入进程的目的，就是为了使多个程序能并发执行
   2) 在引入进程后，若分别为计算程序和 I/O 程序各建立一个进程，则这两个进程便可并发执行。由于在系统中具备使计算程序和 I/O 程序同时运行的硬件条件，因而可将系统中的 CPU 和 I/O 设备同时开动起来，实现并行工作，从而有效地提高了系统资源的利用率和 系统吞吐量，并改善系统的性能
   3) 可以在内存中存放多个用户程序，分别为它们建立进程后，这些进程可以并发执行，亦即实现前面所说的多道程序运行。这样便能极大地提高系统资源的利用率，增加系统的吞吐量。
   4) 进程是**在系统中能独立运行并作为资源分配的基本单位**，由机器指令、数据和堆栈等组成的，活动实体。多进程间可并发执行和交换信息。进程在运行时需要一定的资源，如 CPU、存储空间及 I/O 设备等
3. 引入线程(Threads)
   1) 长期以来，进程都是操作系统中可以拥有资源并独立运行的基本单位。当一个进程因故不能继续运行时，操作系统便调度另一进程运行。由于进程拥有自己的资源，故使调度付出的开销较大。直到 20 世纪 80 年代中期，人们才又提出了比进程更小的单位——线程
   2) 通常在一个进程中可以包含若干个线程，它们可以利用进程所拥有的资源。在引入线程的 OS 中，通常都是**把进程作为分配资源的基本单位，而把线程作为独立运行和独立调度的基本单位。**由于线程比进程更小，基本上不拥有系统资源，故对它的调度所付出的开销就会小得多，能更高效地提高系统内多个程序间并发执行的程度。因而近年来推出的通用操作系统都引入了线程，以便进一步提高系统的并发性，并把它视作现代操作系统的一个重要标致

二. 共享性Sharing

1) 共享：系统中的资源可供内存中多个并发执行的进程(线程)共同使用，相应地，把这种资源共同使用称为资源共享，或称为资源复用。由于各资源的属性不同，进程对资源复用的方式也不同，目前主要实现资源共享的方式有如下两种

1. 互斥共享方式
   1) 互斥式共享：当一个进程 A 要访问某资源时，必须先提出请求。如果此时该资源空闲，系统便可将之分配给请求进程 A 使用。此后若再有其它进程也要访问该资源时(只要 A 未用完)，则必须等待。仅当 A 进程访问完并释放该资源后，才允许另一进程对该资源进行访问
   2) **临界资源/独占资源：在一段时间内只允许一个进程访问的资源，**它们要求被互斥地共享
   3) 计算机系统中的大多数物理设备，以及某些软件中所用的栈、变量和表格，都属于临界资源
2. 同时访问方式
   1) 另一类资源，允许在一段时间内由多个进程“同时”对它们进行访问。这里所谓的“同时”，在单处理机环境下往往是宏观上的，而在微观上，这些进程可能是交替地对该资源进行访问
   2) 如磁盘设备，一些用重入码编写的文件也可以被“同时”共享，即若干个用户同时访问该文件
3. 并发和共享是操作系统的两个最基本的特征，互为存在的条件。一方面，资源共享以程序(进程)的并发执行为条件，不允许程序并发执行，自然不存在资源共享问题；另一方面，不能有效

管理资源共享，协调好诸进程对共享资源的访问，也必然影响到并发执行的程度，甚至根本无法并发执行

三. 虚拟技术Virtual

1) 虚拟：通过某种技术把一个物理实体变为若干个逻辑上的对应物。物理实体(前者)是实际存在的，而后者是虚的，仅是用户感觉上的东西

1. 时分复用技术/分时使用方式：提高资源利用率。最早用于电信业中。为了提高信道的利用率，人们利用时分复用方式，将一条物理信道虚拟为多条逻辑信道，将每条信道供一对用户通话
   1) 虚拟处理机技术：利用多道程序设计技术，为每道程序建立一个进程，让多道程序并发地执行，以此来分时使用一台处理机。虽然系统中只有一台处理机，但它能同时为多个用户服务，使每个终端用户都认为是有一个处理机在专门为他服务。亦即，利用多道程序设计技术，把一台物理上的处理机虚拟为多台逻辑上的处理机，在每台逻辑处理机上运行一道程序。把用户所感觉到的处理机称为虚拟处理器
   2) 虚拟设备技术：将一台物理 I/O 设备虚拟为多台逻辑上的 I/O 设备，并允许每个用户占用一台逻辑上的 I/O 设备，这样便可使原来仅允许在一段时间内由一个用户访问的设备(即临界资源)，变为在一段时间内允许多个用户同时访问的共享设备。例如，原来的打印机属于临界资源，而通过虚拟设备技术，可以把它变为多台逻辑上的打印机，供多个用户“同时”打印。关于虚拟设备技术将在第五章中介绍
2. 空分复用技术：提高存储空间的利用率。电信业将一个频率范围非常宽的信道，划分成多个频率范围较窄的信道，其中的任何一个频带都只供一对用户通话
   1) 虚拟磁盘技术：将一台硬盘虚拟为多台虚拟磁盘
      1) 将硬盘划分为若干个卷，例如 1、2、3、4 四个卷，再通过安装程序将它们分别安装在 C、 D、E、F 四个逻辑驱动器上，这样，机器上便有了四个虚拟磁盘
   2) 虚拟存储器技术：用存储器的空闲空间存放程序，提高内存的利用率
      (1) 单纯的空分复用存储器只能提高内存的利用率，并不能实现在逻辑上扩大存储器容量的功能，必须引入虚拟存储技术才能达到此目的。而虚拟存储技术在本质上就是使内存分时复用。它可以使一道程序通过时分复用方式，在远小于它的内存空间中运行
      (2) 设 N 是某物理设备所对应的虚拟的逻辑设备数，时分复用的每台虚拟设备的平均速度必然<=物理设备速度的 1/N；空分复用每台虚拟设备平均占用的空间必然也<=物理设备所拥有空间的 1/N

四. 异步性Asynchronism

1. 多道程序环境下允许多个进程并发执行，只有进程在获得所需的资源后方能执行。在单处理机环境下，由于系统中只有一台处理机，因而每次只允许一个进程执行，其余进程只能等待。由于资源等因素的限制，使进程的执行通常都不是“一气呵成”，而是以“停停走走”的方式运行
2. 异步性：进程以人们**不可预知的速度向前推进**。只要在操作系统中配置有完善的进程同步机制，且运行环境相同，作业经多次运行都会**获得完全相同的结果**。因此，异步运行方式是允许的，而且是操作系统的一个重要特征

i.
ii.
iii.
iv.
v.
vi. --------我是底线----------

1功能、设计

2018年10月31日 13:36

-
- 操作系统的主要功能

一. 处理机管理功能

1) 创建和撤消进程(线程)，对诸进程(线程)的运行进行协调，实现进程(线程) 之间的信息交换，以及按照一定的算法把处理机分配给进程(线程)

1. 进程控制：主要功能是为作业创建进程，撤消已结束的进程，以及控制进程在运行过程中的状态转换。在现代 OS 中，进程控制还应具有为一个进程创建若干个线程的功能和撤消(终止)已完成任务的线程的功能
2. 进程同步：为多个异步进程(含线程)的运行进行协调。有两种方式：
   1) 进程互斥方式：指访问临界资源时互斥
   2) 进程同步方式：指在相互合作去完成共同任务的诸进程间，由同步机构对它们的执行程序加以协调
      (1) 最简单的实现机制是为每一个临界资源配置一把锁 W，当锁打开时，进程(线程)可以对该临界资源进行访问；而当锁关上时，则禁止进程(线程)访问该临界资源
      (2) 最常用机制是信号量机制
3. 进程通信：相互合作的进程之间的信息交换
   1) 同一计算机系统内，通常是采用直接通信方式，即由源进程利用发送命令直接将消息(Message)挂到目标进程的消息队列上，以后由目标进程利用接收命令从其消息队列中取出消息
4. 调度
   1) 作业调度
      (1) 从后备队列中按照一定的算法，选择出若干个作业，为它们分配运行所需的资源(首先是分配内存)。在后备队列上等待的每个作业都需经过调度才能执行
      (2) 调入内存后，便分别为它们建立进程，使它们都成为可能获得处理机的就绪进程，并按一定的算法将它们插入就绪队列
   2) 进程调度
      (1) 从进程的就绪队列中，按照一定的算法选出一个进程，把处理机分配给它，并为它设置运行现场，使进程投入执行
      (2) 多线程 OS 中，通常把线程作为独立运行和分配处理机的基本单位。把就绪线程排成一个队列，每次调度时，从队列中选出一线程，分配处理机给它

二. 存储器管理功能

1) 提高存储器的利用率以及从逻辑上扩充内存

1. 内存分配：为每道程序分配内存空间，使它们“各得其所”；提高存储器利用率，以减少不可用的内存空间；允许正在运行的程序申请附加的内存空间，以适应程序和数据动态增长的需要
   1) 静态分配方式：每个作业的内存空间是在作业装入时确定的；在作业装入后的整个运行期间，不允许该作业再申请新的内存空间，也不允许作业在内存中“移动”
   2) 动态分配方式：每个作业所要求的基本内存空间也是在装入时确定的，但允许作业在运

行过程中继续申请新的附加内存空间，以适应程序和数据的动态增长，也允许作业在内存中“移动”

3) 内存分配机制里需要的结构和功能
   (1) 内存分配数据结构：记录内存空间的使用情况，作为分配依据
   (2) 内存分配功能：按一定的内存分配算法为用户程序分配内存空间
   (3) 内存回收功能：对于用户不再需要的内存，通过用户的释放请求去完成系统的回收功能

2. 内存保护：确保每道用户程序都只在自己的内存空间内运行，彼此互不干扰；绝不允许用户程序访问操作系统的程序和数据；也不允许用户程序转移到非共享的其它用户程序中去执行
   1) 一种比较简单的内存保护机制是设置两个界限寄存器，分别用于存放正在执行程序的上界和下界，系统须对每条指令所要访问的地址进行检查，如果发生越界，便发出越界中断请求，以停止该程序的执行。若用软件实现，则每执行一条指令，须增加若干条指令去进行越界检查，这将显著降低程序的运行速度。因此，越界检查都由硬件实现，对发生越界后的处理，还须与软件配合来完成

3. 地址映射
   1) 地址空间：地址所形成的地址范围，其中的地址称为“逻辑地址”或“相对地址”。程序的逻辑地址都从“0”开始的，程序中的其它地址都是相对于起始地址计算的
   2) 内存空间：内存中的一系列单元所限定的地址范围，其中的地址称为“物理地址”
   3) 为使程序能正确运行，存储器管理必须提供地址映射功能，以将地址空间中的逻辑地址转换为内存空间中与之对应的物理地址。 该功能应在硬件的支持下完成

4. 内存扩充：借助于虚拟存储技术，从逻辑上去扩充内存容量
   1) 请求调入功能：允许在装入一部分用户程序和数据的情况下，便能启动该程序运行。 在程序运行过程中，若发现要继续运行时所需的程序和数据尚未装入内存，可向 OS 发出请求，由 OS 从磁盘中将所需部分调入内存，以便继续运行
   2) 置换功能：若发现在内存中已无足够的空间来装入需要调入的程序和数据时，系统应能将内存中的一部分暂时不用的程序和数据调至盘上，以腾出内存空间，然后再将所需调入的部分装入内存

三. 设备管理功能

1) 完成用户进程提出的 I/O 请求；为用户进程分配其所需的 I/O 设备；提高 CPU 和 I/O 设备的利用率；提高 I/O 速度；方便用户使用 I/O 设备

1. 缓冲管理
   1) 在 I/O 设备和 CPU 之间引入缓冲，可有效地缓和 CPU 与 I/O 设备速度不匹配的矛盾，提高 CPU 的利用率， 进而提高系统吞吐量
   2) 最常见的缓冲区机制有单缓冲、双向同时传送的双缓冲，供多设备同时使用的公用缓冲池

2. 设备分配
   1) 根据用户进程的 I/O 请求、系统现有资源情况及某种设备的分配策略，为之分配其所需的设备
   2) 如果在 I/O 设备和 CPU 之间还存在着设备控制器和 I/O 通道，还须为分配出去的设备分配相应的控制器和通道
   3) 为了实现设备分配，系统中应设置设备控制表、控制器控制表等数据结构，用于记录设备及控制器的标识符和状态
   4) 在进行设备分配时，应针对不同的设备类型而采用不同的设备分配方式。对于独占设备(临界资源)，还应考虑到该设备被分配出去后系统是否安全。 在设备使用完后，应立即由系统回收

3. 设备处理
   1) 设备处理程序又称为设备驱动程序。其基本任务是用于实现 CPU 和设备控制器之间的通信，即由 CPU 向设备控制器发出 I/O 命令，要求它完成指定的 I/O 操作；反之，由 CPU 接收从控制器发来的中断请求，并给予迅速的响应和相应的处理

2) 处理过程是：设备处理程序首先检查 I/O 请求的合法性，了解设备状态是否空闲，了解有关的传递参数及设置设备的工作方式。然后向设备控制器发出 I/O 命令，启动 I/O 设备完成指定的 I/O 操作。设备驱动程序还应能及时响应由控制器发来的中断请求，并根据该中断请求的类型，调用相应的中断处理程序进行处理。对于设置了通道的计算机系统，设备处理程序还应能根据用户的 I/O 请求，自动地构成通道程序

四. 文件管理功能

1) 对用户文件和系统文件进行管理，方便用户使用，保证文件的安全性

1. 文件存储空间的管理：由文件系统对诸多文件及存储空间实施统一的管理。其主要任务是为每个文件分配必要的外存空间，提高外存的利用率，并能有助于提高文件系统的存、取速度

1) 系统应设置相应的数据结构，记录文件存储空间的使用情况，以供分配存储空间时参考；还应具有对存储空间进行分配和回收的功能。为了提高存储空间的利用率，对存储空间的分配，通常是采用离散分配方式，以减少外存零头，并以盘块为基本分配单位。盘块的大小通常为 1～8 KB。

2. 目录管理

1) 由系统为每个文件建立一个目录项，包括文件名、文件属性、文件在磁盘上的物理位置等
2) 由若干个目录项又可构成一个目录文件
3) 主要任务是为每个文件建立其目录项，并对众多的目录项加以有效的组织，以实现方便的按名存取，即用户只须提供文件名便可对该文件进行存取
4) 还应能实现文件共享，这样，只须在外存上保留一份该共享文件的副本
5) 还应能提供快速的目录查询手段，以提高对文件的检索速度

3. 文件的读/写管理和保护

1) 文件的读/写管理：该功能是根据用户的请求，从外存中读取数据，或将数据写入外存。在进行文件读(写)时，系统先根据用户给出的文件名去检索文件目录，从中获得文件在外存中的位置。然后，利用文件读(写)指针，对文件进行读(写)。一旦读(写)完成，便修改读(写)指针，为下一次读(写)做好准备。由于读和写操作不会同时进行，故可合用一个指针

2) 文件保护：为防止文件被非法窃取和破坏，必须提供有效的存取控制功能，以实现下述目标：

(1) 防止未经核准的用户存取文件
(2) 防止冒名顶替存取文件
(3) 防止以不正确的方式使用文件

五. 操作系统与用户之间的接口

1. 用户接口：提供给用户使用的接口，用户可通过该接口取得操作系统的服务

1) 联机用户接口：这是为联机用户提供的，它由一组键盘操作命令及命令解释程序所组成。当用户在终端或控制台上每键入一条命令后，系统便立即转入命令解释程序，对该命令加以解释并执行该命令。在完成指定功能后，控制又返回到终端或控制台上，等待用户键入下一条命令。这样，用户可通过先后键入不同命令的方式，来实现对作业的控制，直至作业完成

2) 脱机用户接口/批处理用户接口：该接口是为批处理作业的用户提供的，由一组作业控制语言(JCL)组成。批处理作业的用户不能直接与自己的作业交互作用， 只能委托系统代替用户对作业进行控制和干预。JCL便是提供给批处理作业用户的、为实现所需功能而委托系统代为控制的一种语言。用户用 JCL 把需要对作业进行的控制和干预事先写在作业说明书上，然后将作业连同作业说明书一起提供给系统。 当系统调度到该作业运行时，又调用命令解释程序，对作业说明书上的命令逐条地解释执行，直至遇到作业结束语句时，系统才停止该作业的运行

3) 图形用户接口：用户可用鼠标或通过菜单和对话框来完成对应用程序和文件的操作。用户从繁琐且单调的操作中解脱了出来。 图形用户接口方便地将文字、图形和图像集成在一个文件中。可以在文字型文件中加入一幅或多幅彩色图画，也可以在图画中写入必要的文字，而且还可进一步将图画、文字和声音集成在一起。20 世纪 90 年代以后推出的主流 OS 都提供了图形用户接口

2. 程序接口：提供给程序员在编程时使用的接口，是用户程序取得操作系统服务的唯一途径

1) 由一组系统调用组成，每个系统调用都是一个能完成特定功能的子程序，每当应用程序要求 OS 提供某种服务(功能)时，便调用具有相应功能的系统调用。早期的系统调用都是汇编语言提供的，只有在用汇编语言书写的程序中才能直接使用系统调用；但在高级语言以及 C 语言中，往往提供了与各系统调用一一对应的库函数，这样，应用程序便可通过调用对应的库函数来使用系统调用。但在近几年所推出的操作系统中，如 UNIX、OS/2 版本中，其系统调用本身已经采用 C 语言编写，并以函数形式提供，故在用 C 语言编制的程序中，可直接使用系统调用

六. 现代操作系统的新功能

1. 系统安全

1) 认证技术：通过一些参数的真实性来确认对象是否名副其实
2) 密码技术：为存储和传输的数据加密
3) 访问控制技术：设置用户存取权限、设置文件属性
4) 反病毒技术：预防、及时扫描可执行文件，清除病毒

2. 网络功能和服务

1) 网络通信：建立和拆除通信链路、传输控制、差错控制、流量控制等
2) 资源管理：协调硬、软件共享资源
3) 应用互操作：互联网络中实现信息的互通性和消息的互用性

3. 支持多媒体

1) 接纳控制功能
   (1) 限制软实时任务数、驻留在内存中的任务数
   (2) 媒体服务器、存储器、进程都设置了相应的接纳控制功能
2) 实时调度
   (1) 考虑进程调度策略的同时还要考虑进程调度的接纳度
   (2) 如图像更新周期应在40ms内
3) 多媒体文件的存储
   (1) 尽量把硬盘上的数据快速转到输出设备上
   (2) 因而需要改进数据离散存放方式和磁盘寻道方式

-
- OS结构设计

一. 传统的操作系统结构

1. 无结构操作系统/整体系统结构

1) 早期设计者只把注意力放在功能的实现和获得高的效率上，此时的 OS 是为数众多的一组过程的集合，每个过程可以任意地相互调用其它过程，致使操作系统内部既复杂又混乱，调试工作带来很多困难；另一方面也使程序难以阅读和理解，增加了维护人员的负担

2. 模块化结构OS

1) 模块化程序设计技术是 20 世纪 60 年代出现的一种结构化程序设计技术。该技术是基于“分解”和“模块化”原则来控制大型软件的复杂度。使各模块之间能通过该接口实现交互。然后，再进一步将各模块细分为若干个具有一定功能的子模块，这种设计方法称为模块—接口法

2)

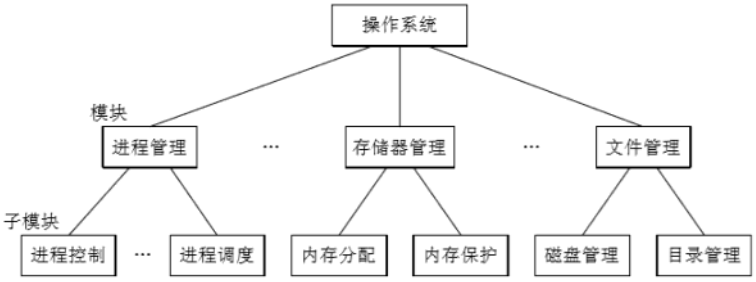


图 1-6 模块化结构的操作系统

3) 模块的独立性
   (1) 内聚性：指模块内部各部分间联系的紧密程度。
      i. 内聚性越高，模块的独立性越强

(2) 耦合度：指模块间相互联系和相互影响的程度。

i. 耦合度越低，模块的独立性越好

4) 模块接口法/无序模块法的优缺点

(1) 提高 OS 设计的正确性、可理解性和可维护性

(2) 增强 OS 的适应性

(3) 加速 OS 的开发过程

(1) 在 OS 设计时，对各模块间的接口规定很难满足在模块完成后对接口的实际需求

(2) 在 OS 设计阶段，设计者必须做出一系列的决定(决策)，每一个决定必须建立在上一个决定的基础上。但在模块化结构设计中，各模块的设计齐头并进，造成各种决定的“无序性”

3. 分层式结构OS

1) 自底向上的分层设计的基本原则是：每一步设计都是建立在可靠的基础上， 每一层仅能使用其底层所提供的功能和服务，每层又由若干个模块组成， 各层之间只存在着单向的依赖关系，即高层仅依赖于紧邻它的低层

2) 分层结构的优缺点

(1) 易保证系统的正确性

(2) 易扩充和易维护性，不修改接口就不影响其他层

(1) 系统效率低，增加了系统的通信开销

二. 客户/服务器模式(Client/Server)

1. 客户/服务器模式模式的组成

1) 客户机：可发送一个消息给服务器，以请求某项服务，平时它处理一些本地业务

2) 服务器：驻留有网络文件系统或数据库系统等， 它应能为网上所有的用户提供一种或多种服务。平时它一直处于工作状态，被动地等待来自客户机的请求，并将结果送回客户

3) 网络系统：用于连接所有客户机和服务器，实现它们之间通信和网络资源共享的系统

2. 客户/服务器之间的交互

1) 客户发送请求消息：发送进程先把命令和有关参数装配成请求消息，然后把它发往服务器；接收进程则等待接收从服务器发回来的响应消息

2) 服务器接收消息：接收进程平时处于等待状态，根据请求信息的内容，将之提供给服务器上的相应软件进行处理

3) 服务器回送消息：软件根据请求进行处理，在完成指定的处理后，把处理结果装配成一个响应消息，由发送进程发往客户机

4) 客户机接收消息：接收进程把收到的响应消息转交给软件，再由后者做出适当处理后提交给发送该请求的客户

3. 客户/服务器模式的优缺点

1) 数据的分布处理和存储

2) 便于集中管理

3) 灵活性和可扩充性

4) 易于改编应用软件

1) 存在着不可靠性和瓶颈问题。一旦服务器故障，将导致整个网络瘫痪。当服务器在重负荷下工作时，会因忙不过来而显著地延长对用户请求的响应时间。如果在网络中配置多个服务器，并采取相应的安全措施，则可加以改善

三. 面向对象的程序设计

1. “重用”提高产品质量和生产率； 具有更好的易修改性和易扩展性；更易于保证系统的“正确性”和“可靠性”

四. 微内核OS结构

1. 微内核操作系统的基本概念

(1) 为了提高操作系统的正确性、灵活性、易维护性和可扩充性：

1) 足够小的内核：

(1) 实现与硬件紧密相关的处理

(2) 实现一些较基本的功能

(3) 负责客户和服务器之间的通信

2) 基于客户/服务器模式

(1) 最基本的部分放入内核中，而把绝大部分功能都放在微内核外面的一组服务器(进程)中实现

(2)

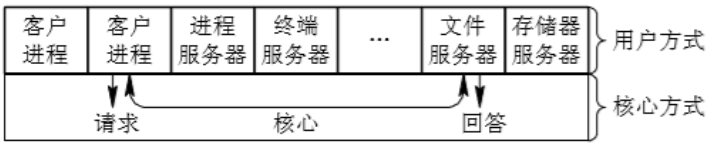


图 1-10 在单机环境下的客户/服务器模式

3) 应用“机制与策略分离”原理

1) 机制，是指实现某一功能的具体执行机构
2) 策略，是在机制的基础上，借助于某些参数和算法来实现该功能的优化
3) 通常，机制处于系统的基层（微内核），而策略则处于高层

4) 采用面向对象技术

(1) “抽象”和“隐蔽”原则控制系统的复杂性，“对象”、“封装”和“继承”等概念来确保操作系统的“正确性”、“可靠性”、“易修改性”、“易扩展性”等

2. 微内核的基本功能

1) 进程(线程)管理：在进程管理中设置一个或多个进程(线程)优先级队列； 能将指定优先级进程(线程)从所在队列中取出，并将其投入执行。由于这一部分属于调度功能的机制部分，应将它放入微内核中。应如何确定每类用户(进程)的优先级，以及应如何修改它们的优先级等，都属于策略问题，可将它们放入微内核外的进程(线程)管理服务器中。

由于进程(线程)之间的通信功能是微内核 OS 最基本的功能，被频繁使用，因此几乎所有的微内核 OS 都是将进程(线程)之间的通信功能放入微内核中。此外,还将进程的切换、线程的调度，以及多处理机之间的同步等功能也放入微内核中

2) 低级存储器管理：如用于实现将用户空间的逻辑地址变换为内存空间的物理地址的页表机制和地址变换机制，这部分依赖于机器，因此放入微内核。而实现虚拟存储器管理的策略，则包含应采取何种页面置换算法，采用何种内存分配与回收策略等，应放在微内核外的存储器管理服务器中

3) 中断和陷入处理：主要功能是捕获所发生的中断和陷入事件，并进行相应的前期处理。如进行中断现场保护，识别中断和陷入的类型，然后将有关事件的信息转换成消息后，把它发送给相关的服务器。由服务器根据中断或陷入的类型，调用相应的处理程序来进行后期处理。 在微内核 OS 中是将进程管理、存储器管理以及 I/O 管理这些功能一分为二，属于机制的很小一部分放入微内核中

3. 微内核操作系统的优点

1) 提高了系统的可扩展性（由于微内核 OS 许多功能是由相对独立的服务器软件实现的）

2) 增强了系统的可靠性

(1) 微内核是出于精心设计和严格测试的容易保证其正确性
(2) 提供了规范而精简的应用程序接口(API)，为微内核外的高质量程序代码创造了条件
(3) 所有服务器都是运行在用户态，服务器与服务器之间采用的是消息传递通信机制，服务器出现错误时，不会影响内核，也不会影响其它服务器

3) 可移植性（所有与特定 CPU 和 I/O 设备硬件有关的代码，均放在内核和内核下面的硬件隐藏层中， 而操作系统其它绝大部分均与硬件平台无关）

4) 提供了对分布式系统的支持

(1) 客户和服务器之间以及服务器和服务器之间的通信，是采用消息传递通信机制进行的，致使微内核 OS 能很好地支持分布式系统和网络系统
(2) 只要在分布式系统中赋予所有进程和服务器惟一的标识符，在微内核中再配置一张系统映射表 (进程和服务器的标识符与它们所驻留的机器之间的对应表)，在进行客户与服务器通信时，只需在所发送的消息中标上发送进程和接收进程的标识符，微内核便可发送消息

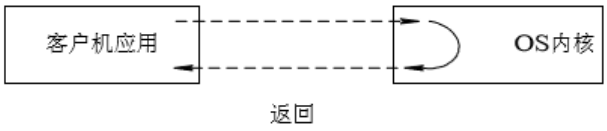
5) 融入了面向对象技术

(1) “封装”，“继承”，“类”和“多态”，以及对象间采用消息传递机制等，都有利于提高系统“正确性”、“可靠性”、“易修改性”、“易扩展性”等，还能显著减少开发系统所付出的开销

4. 微内核操作系统存在的问题

1) 运行效率有所降低：在完成一次客户对 OS 提出的服务请求时，需要利用消息实现多次交互

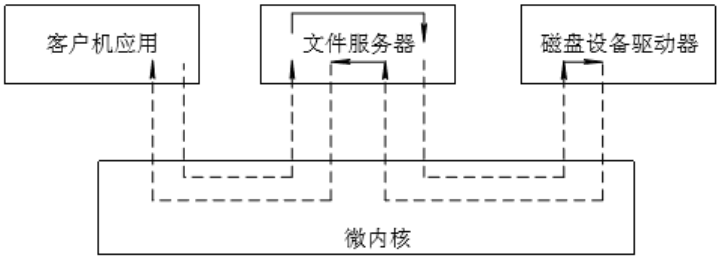
和进行用户/内核模式及上下文的多次切换。然而，在早期的 OS 中，用户进程在请求取得 OS 服务时，一般只需进行两次上下文的切换。如图 1-11 中所示，其中的文件服务器还需要磁盘服务器的帮助，这时就需要进行八次上下文的切换



返回

(a) 在整体式内核文件操作中的上下文切换

2)



(b) 在微内核中等价操作的上下文切换

3) 为改善运行效率，可以重新把一些常用的基本功能由服务器移入微内核中。但这又会使微内核的容量明显地增大，在小型接口定义和适应性方面的优点也有所下降，也提高了微内核的设计代价

i.
ii.
iii.
iv.
v.
vi. --------我是底线----------

2进程描述、控制

2018年10月8日 8:00

-
- 前趋图和程序执行

1. 操作系统四大特征都是基于进程而形成的

一. 前趋图

1. 前趋图Precedence Graph
   1) 有向无循环图Directed Acyclic Graph
   2) 每个结点可用于描述一个程序段或进程，乃至一条语句
   3) 有向边表示两个结点之间存在的偏序关系(Partial Order) 或前趋关系(Precedence Relation)“→”
   4) (Pi，Pj)∈→，可写成 Pi→Pj
2. 结点
   (1) 把没有前趋的结点称为初始结点(Initial Node)
   (2) 把没有后继的结点称为终止结点(Final Node)
   1) 每个结点有一个重量(Weight)，表示该结点所含有的程序量或执行时间

二. 程序顺序执行

1. 顺序执行：仅当前一操作(程序段)执行完后，才能执行后继操作
2. 程序顺序执行时的特征
   (1) 顺序性：处理机严格按程序规定顺序执行操作：每一操作必须在上一操作结束后开始
   (2) 封闭性：程序运行时独占全机资源，资源的状态(除初始状态外)只有本程序才能改变。程序一旦开始执行，其执行结果不受外界因素影响
   (3) 可再现性：只要程序执行时的环境和初始条件相同，当程序重复执行时，不论它是从头到尾不停顿地执行，还是“停停走走”地执行，都将获得相同的结果

三. 程序并发执行

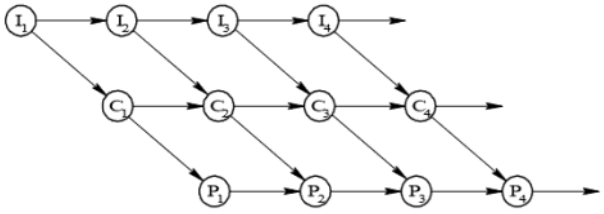
1. 

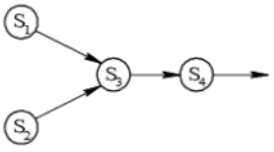
图 2-3 并发执行时的前趋图
   1) 在该例中存在下述前趋关系：Ii→Ci，Ii→Ii+1，Ci→Pi，Ci→Ci+1，Pi→Pi+1
   2) 而 Ii+1和 Ci及 Pi-1是重迭的，在 Pi-1和 Ci以 及 Ii+1 之间，可以并发执行
2. 

图 2-4 四条语句的前趋关系
   1) S3必须在 a 和 b 被赋值后方能执行
   2) S4必须在 S3之后执行
   3) S1和 S2则可以并发执行，因为它们彼此互不依赖
3. 程序并发执行时的特征
   1) 间断性：并发执行的程序之间，有相互制约的关系，导致“执行—暂停—执行”的活动规律
   2) 失去封闭性：共享资源的状态将由多个程序来改变。这样，某程序在执行时，必然会受到其它程序的影响。例如，当处理机这一资源已被某个程序占有时，另一程序必须等待。

3) 不可再现性：由于失去了封闭性，执行时的环境和初始条件相同，但得到的结果却不相同

-
- 进程的描述

一. 进程的定义和特征

1. 进程控制块(Process Control Block)

1) 为使程序(含数据)能独立运行，应为之配置一进程控制块

2) 进程实体/进程映像：由程序段、相关的数据段和 PCB 三部分构成

3) 创建进程实质上是创建PCB；而撤消进程实质上是撤消PCB

2. 进程的定义

1) 进程是程序的一次执行

2) 进程是一个程序及其数据在处理机上顺序执行时所发生的活动

3) 进程是程序在一个数据集合上运行的过程，它是系统进行资源分配和调度的一个独立单位

4) 进程是进程实体的运行过程，是系统进行资源分配和调度的独立单位

3. 进程的特征

1) 动态性

(1) 进程的实质是进程实体的一次执行过程，因此，动态性是进程的最基本的特征

(2) 由创建而产生，由调度而执行，由撤消而消亡

(3) 进程实体有生命期，而程序则只是一组存放于某种介质上有序指令的集合，是静态的

2) 并发性：多个进程实体同存于内存中，且能在一段时间内同时运行

3) 独立性：独立运行、独立分配资源和独立接受调度

4) 异步性：进程按各自独立的、 不可预知的速度向前推进，或说进程实体按异步方式运行

(1) 应通过配置相应的进程同步机制，保证进程并发执行的结果是可再现的

1. 进程和作业

1) 进程总在内存，作业刚提交时在外存等待

2) 作业可由多个进程组成，反之不然

3) 作业的概念主要出现在批处理系统中，进程用在几乎所有多道系统中

2. 进程和程序

1) 进程动态，程序静态

2) 程序可对应多个进程，进程可包含多个程序

二. 进程的基本状态及转换

1. 进程的三种基本状态

1) 就绪(Ready)状态

(1) 已分配到**除 CPU 以外的必要资源**，只要获得 CPU，便可立即执行

(2) 通常将它们按一定策略排成一个就绪队列

2) 执行(Running)状态

(1) 进程已获得 CPU，其程序正在执行

(2) 处理机数量决定处于执行状态的进程的数量

3) 阻塞(Block)状态/等待状态/封锁状态

(1) 发生某事件而暂时无法继续执行时，放弃处理机而暂停

(2) 此时引起进程调度，OS把处理机分配给另一个就绪程序

(3) 根据请求 I/O，申请缓冲空间等阻塞原因把处于阻塞状态的进程排成多个队列

4)

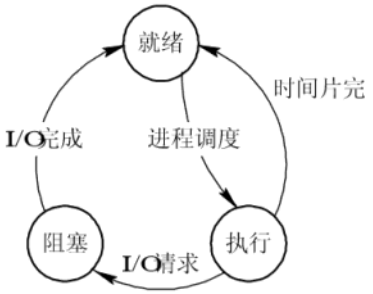


图 2-5　进程的三种基本状态及其转换

2. 创建状态和终止状态

1) 创建状态

(1) 创建过程：创建 PCB，并填写必要的管理信息；分配运行时必要的资源；把该进程转入就绪状态并插入就绪队列之中

(2) ↑若已创建PCB，而资源尚未分配完，此时的状态即为~

(3) 可以根据系统性能或主存容量的限制，推迟创建状态进程的提交

2) 终止状态

(1) 当一个进程到达了自然结束点，或出现了无法克服的错误，或被操作系统所终结，或被其他有终止权的进程所终结，将进入~

(2) 终止过程：善后处理，将其 PCB 清零，并将 PCB 空间返还系统

(3) 进入终止态以后不能再执行，但在操作系统中依然保留状态码和一些计时统计数据，供其它进程收集，一旦其它进程完成了对终止状态进程的信息提取，操作系统将删除该进程

(4)

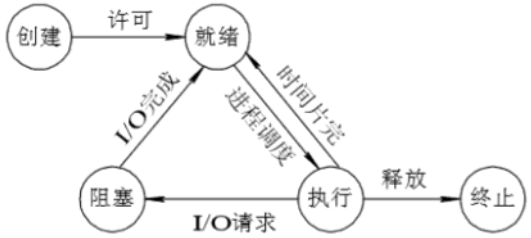


图 2-7　进程的五种基本状态及转换

三. 挂起操作和进程状态的转换

1. 挂起操作的引入原因

1) 终端用户的请求：终端用户在程序运行期间发现有可疑问题时，希望程序暂停执行，以便用户研究其执行情况或对程序进行修改

2) 父进程请求：考查和修改该子进程，或者协调各子进程间的活动

3) 负荷调节的需要：系统把一些不重要的进程挂起，以保证正常运行

4) 操作系统的需要：检查运行中的资源使用情况或进行记账

2. 进程状态的转换

1) 活动就绪→静止就绪

(1) 未被挂起的就绪状态，称为活动就绪， 表示为 Readya

(2) 用挂起原语 Suspend 将该进程挂起后，该进程便转变为静止就绪状态，表示为 Readys

2) 活动阻塞→静止阻塞

(1) 未被挂起的阻塞状态，称活动阻塞状态，表示为 Blockeda

(2) 用 Suspend 原语将它挂起后，进程便转变为静止阻塞状态，表示为 Blockeds。处于该状态的进程在其所期待的事件出现后，将从静止阻塞变为静止就绪

3) 静止就绪→活动就绪：处于 Readys 状态的进程，用激活原语 Active 激活后，转为 Readya 状态
4) 静止阻塞→活动阻塞：处于 Blockeds 状态的进程，用激活原语 Active 激活后，转为 Blockeda 状态

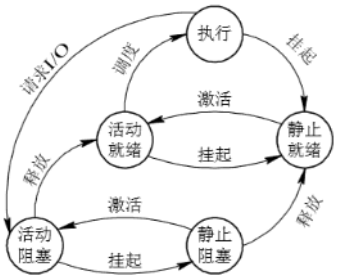


图 2-6 具有挂起状态的进程状态图

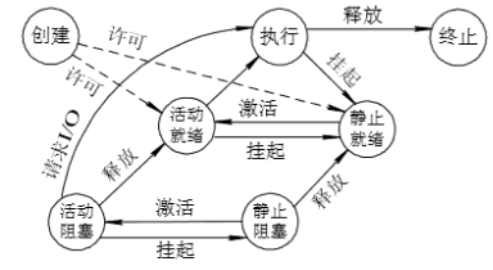


图 2-8 具有创建、终止和挂起状态的进程状态图

四. 进程管理中的数据结构

1. 操作系统中用于管理控制的数据结构

1) 资源信息表/进程信息表：标识、描述、状态等信息和一些指针
2) 内存、设备、文件、进程都有表，其中进程表又被称为进程控制块

2. 进程控制块的作用

1) 作为独立运行基本单位的标识
2) 实现间断性运行的方式（保存cpu现场信息）
3) 提供进程管理所需的信息（保存数据指针、文件设备等资源的访问、资源清单）
4) 提供进程调度所需的信息（保存状态信息、优先级、等待时间、执行时间等）
5) 实现与其他进程的同步与通信（实现进程通信的区域或通信队列指针）

3. 进程控制块中的信息

1) 进程标识符

(1) 内部标识符：为方便调用每一个进程赋予一个惟一的数字标识符，通常是一个序号
(2) 外部标识符：由创建者提供，通常是字母、数字组成，往往是在访问该进程时使用
   i. 为了描述进程的家族关系，还应设置父进程标识及子进程标识
   ii. 还可设置用户标识，以指示拥有该进程的用户

2) 处理机状态/处理机上下文

(1) 通用寄存器/用户可视寄存器：它们是用户程序可以访问的，用于暂存信息
(2) 指令计数器：其中存放了要访问的下一条指令的地址
(3) 程序状态字 PSW：其中含有状态信息，如条件码、执行方式、中断屏蔽标志等
(4) 用户栈指针：存放过程和系统调用参数及调用地址的栈的指针，指向该栈的栈顶

3) 进程调度信息

(1) 进程状态：指明进程的当前状态，作为进程调度和对换时的依据
(2) 进程优先级：一个整数，优先级高的进程应优先获得处理机
(3) 其它信息：与进程调度算法有关，如，等待 CPU 的时间总和、已执行时间总和
(4) 阻塞原因：进程由执行状态转变为阻塞状态所等待发生的事件

4) 进程控制信息

(1) 程序和数据的地址：程序和数据所在地址，以便从 PCB 中找到其程序和数据
(2) 进程同步和通信机制：消息队列指针、信号量等
(3) 资源清单：除 CPU 以外的、进程所需的全部资源及已经分配到该进程的资源的清单
(4) 链接指针：给出本进程(PCB)所在队列中的下一个进程的 PCB 的首地址

4. 进程控制块的组织方式

1) 线性方式：所有PCB组织在一张线性表。只适合进程数目不多的系统
2) 链接方式：把同状态的 PCB用链接字链接成就绪队列、若干阻塞队列和空白队列等

(1)

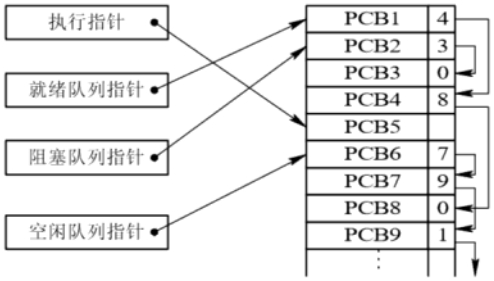


图 2-9 PCB 链接队列示意图

3) 索引方式:根据所有进程的状态建立几张索引表，并把各索引表在内存的首地址记录在内存的一些专用单元中。在每个索引表的表目中，记录具有相应状态的某个 PCB 在 PCB 表中的地址

(1)

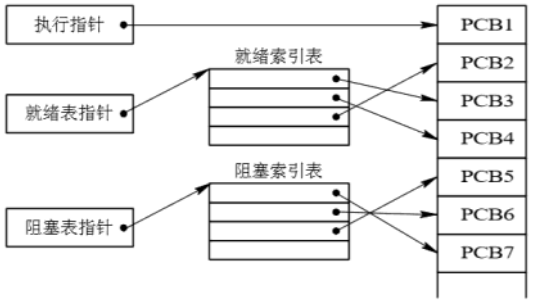


图 2-10 按索引方式组织 PCB

-
- 进程控制

一. 操作系统内核

1) 通常与硬件紧密相关的模块、常用设备的驱动程序、运行频率较高的模块都安排在紧靠硬件的软件层次中，将他们常驻内存，称其OS**内核**
2) 处理机系统态/管态/内核态：有较高特权，能执行一切指令，访问所有寄存器和存储区
3) 用户态/目态：仅能执行规定指令，访问指定寄存器和存储区，是一般情况的执行状态

1. 支撑功能

1) 中断处理：最基本的功能，系统调用、键盘命令的输入、进程调度、设备驱动等都需要它
2) 时钟管理：如时间片结束后，时钟管理会产生一个中断信号
3) 原语操作：链表操作、进程同步等
   (1) 原语Primitive：若干条指令组成的，有一定功能的一个过程
      i. 是原子操作Action Operation：不可分割
      ii. 要么全做，要么不做，不能被中段
      iii. 在系统态下执行，常驻内存

2. 资源管理功能

1) 进程管理：调度分派、创建撤销、同步通信等
2) 存储器管理：逻辑/物理地址转换、内存分配回收、内存保护对换等
3) 设备管理：设备驱动程序、缓冲管理、设备分配和独立性功能模块等

二. 进程的创建

1. 进程的层次结构：父进程、子进程、孙进程组成的进程家族
2. 进程图Process Graph

1) 进程图是描述进程家族关系的有向树
2) 结点(圆圈)代表进程
3) 进程 D 创建了进程 I，称 D 是 I 的父进程Parent Process，I 是 D 的子进程Progeny Process

4) 创建父进程的进程是祖先进程，树的根节点是进程家族的祖先Ancestor

5) 由父进程指向子进程的有向边描述了父子关系

6) 子进程可以继承父进程的资源，如，父进程打开的文件，父进程所分配到的缓冲区等

7) 子进程被撤消时，应将继承的的资源归还给父进程。撤消父进程时，应同时撤消其所有的子进程

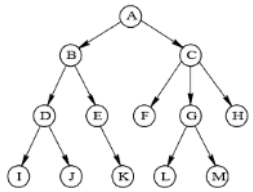
8) 

图 2-11 进程树

3. 引起创建进程的事件

1) 系统内核：

(1) 用户登录：分时系统合法用户登陆后，该终端得到一个进程，并被插入就绪队列中

(2) 作业调度：批处理系统调度到某作业时，将为该作业创建进程，再插入就绪队列中

(3) 提供服务：运行中的用户程序提出某请求后，系统将专门创建一个进程来提供服务

2) 应用请求：进程自己创建一个新进程，以便以并发运行方式完成特定任务，如IO

4. 进程的创建(Creation of Process) 即原语 Creat( )内容：

1) 申请空白 PCB：为新进程申请获得惟一的数字标识符，并从 PCB 集合中索取一个空白 PCB

2) 为新进程分配资源：为新进程的程序和数据以及用户栈分配必要的内存空间

3) 初始化进程控制块

(1) 初始化标识信息，将系统分配的标识符和父进程标识符填入新 PCB 中

(2) 初始化处理机状态信息，使程序计数器指向程序的入口地址，使栈指针指向栈顶

(3) 初始化处理机控制信息，将进程的状态设置为就绪状态或静止就绪状态，对于优先级，通常是将它设置为最低优先级，除非用户以显式方式提出高优先级要求

(4) 将新进程插入就绪队列，如果就绪队列能够接纳新进程，便将新进程插入

三. 进程的终止

1. 引起进程终止的事件

1) 正常结束:用于表示进程已经运行完成的指示。批处理系统中，通常在程序的最后安排一条 Halt 指令或终止的系统调用。在分时系统中，用户可利用 Logs off 去表示进程运行完毕

2) 异常结束：某些错误和故障迫使进程终止(Termination of Process)

(1) 越界错误：程序所访问的存储区已越出该进程的区域

(2) 保护错：进程试图去访问一个不允许访问的资源或文件，或者以不适当的方式进行访问，如，写一个只读文件

(3) 非法指令：试图去执行不存在的指令。可能是程序错误地转移到数据区，把数据当成了指令

(4) 特权指令错：用户进程试图去执行一条只允许 OS 执行的指令

(5) 运行超时：进程的执行时间超过了指定的最大

(6) 等待超时：进程等待某事件的时间超过了规定的最大值

(7) 算术运算错：进程试图去执行一个被禁止的运算，如被 0 除

(8) I/O 故障：I/O 过程中发生了错误等

3) 外界干预：进程应外界的请求而终止运行

(1) 操作员或操作系统干预：如，发生了死锁，由操作员或操作系统 终止该进程

(2) 父进程请求：父进程有终止子孙进程的权力。父进程提出请求时，系统将终止该进程

(3) 父进程终止：当父进程终止时，OS 也将它的所有子孙进程终止

2. 进程的终止过程

1) 读出进程状态（根据标识符从 PCB 集合中检索其 PCB）
2) 终止执行，并置调度标志为真，用于指示该进程应重新调度
3) 终止所有子孙进程，以防它们成为不可控的进程
4) 资源归还给其父进程或系统
5) 移出PCB（从所在队列(或链表)），等待其他程序来搜集信息

3. 进程的阻塞与唤醒

1) 引起进程阻塞和唤醒的事件：
   (1) 请求系统资源失败：如打印机，仅在其他进程在释放出打印机的同时，才将请求进程唤醒
   (2) 等待某种操作：如进程在启动了 I/O 操作后，便自动进入阻塞状态，在 I/O 操作完成后，再由中断处理程序或中断进程将该进程唤醒
   (3) 新数据尚未到达：如 A 尚未将数据输入完毕，则进程 B 将因没有所需的处理数据而阻塞；一旦进程 A 把数据输入完毕，便可去唤醒进程 B
   (4) 无新任务到达：如，系统中的发送进程，已有的数据已全部发送完成而又无新的发送请求，这时发送进程将使自己进入阻塞状态； 仅当有进程提出新的发送请求时，才唤醒
2) 进程阻塞过程：调用阻塞原语 block 把自己阻塞。可见，阻塞是进程自身的一种主动行为
   (1) 先立即停止执行，把进程控制块中的现行状 态由“执行”改为“阻塞”
   (2) 将 PCB 插入到具有相同事件的阻塞(等待)队列
   (3) 转调度程序进行重新调度，即，在 PCB 中保留被阻塞进程的处理机状态，再按新进程的 PCB 中的处理机状态设置 CPU 的环境
3) 进程唤醒过程：调用唤醒原语 wakeup( )
   (1) 把进程从等待该事件的阻塞队列中移出
   (2) 将其 PCB 中的现行状态由阻塞改为就绪
   (3) 将该 PCB 插入到就绪队列中

四. 进程的挂起与激活

1. 进程的挂起：利用挂起原语 suspend( )将指定进程或处于阻塞状态的进程挂起
   1) 检查进的状态，活动就绪状态，改为静止就绪； 活动阻塞状态改为静止阻塞
   2) 把该进程的 PCB 复制到某指定的内存区域，方便用户或父进程考查该进程的运行情况
   3) 若被挂起的进程正在执行，则转向调度程序重新调度
2. 进程的激活过程：利用激活原语 active( )
   1) 将进程从外存调入内存
   2) 静止就绪改为活动就绪；静止阻塞改为活动阻塞
      (1) 插入就绪队列时假如采用的是抢占调度策略，应由调度程序进行优先级的比较，如果被激活进程的优先级高，立即剥夺当前进程的运行，把处理机分配给刚被激活的进程

进程不可能由自己创建/唤醒/激活/终止。可能由自己挂起/阻塞

i.
ii.
iii.
iv.
v.
vi.
vii. --------我是底线----------

2进程同步

2018年10月31日 13:37

-
- 进程同步

一. 进程同步的基本概念

1. 两种形式的制约关系
   1) **间接相互制约**关系：源于**互斥**访问临界资源
   2) **直接相互制约**关系：源于进程间的**合作**
2. 临界资源：一段时间内只允许一个进程访问的资源
3. 临界区(critical section)：在每个进程中访问临界资源的那段代码
   1) 保证诸进程互斥地进入自己的临界区即可实现对临界资源的互斥访问
   2) 在进入临界区之前，应先对欲访问的临界资源进行检查，如果此刻该临界资源正被某进程访问，则本进程不能进入临界区
   3) 检查临界资源的代码称为进入区(entry section)；恢复临界资源为未被访问的标志的代码称为退出区(exit section)；称进入区临界区退出区以外的代码为剩余区(remainder section)
4. 同步机制应遵循的规则
   1) **空闲让进**：无进程处于临界区时，临界资源处于空闲状态，应允许进程立即进入自己的临界区
   2) **忙则等待**：已有进程进入临界区时，表明临界资源正在被访问，其它进程必须等待
   3) **有限等待**：应保证进程在有限时间内能进入自己的临界区， 以免陷入“死等”状态
   4) **让权等待**：当进程不能进入自己的临界区时，应立即释放处理机，以免进程陷入“忙等”状态

二. 硬件同步机制

1) 用一个标志管理临界区，称其为锁

1. 关中断：进入锁测试前关闭中断，完成锁测试后打开中断
   1) 滥用关中断权力可能导致严重后果
   2) 关中断时间长，影响系统效率，限制处理器交叉执行程序的能力
   3) 不适用于多CPU系统，因为一个处理器上关中断并不能阻止其他处理器
2. Test-and-Set指令/原语
   1) boolean TS(boolean*lock){

```
boolean old=lock;
*lock=TRUE;//无论原值多少，调用了TS后总变为true
return old;}
```

   2) do{

```
while(TS(&lock))  ;
/*critical section*/
lock=FALSE;
/*remainder section*/
}while(TRUE);
```

3. swap指令，如Intel 80X86的XCHG指令
   1) do{

```
key=TRUE;
do{
    swap(&lock,&key);
}while(key!=FALSE);
}while(TRUE);
```

4. 上述都是忙等，不符合让权等待原则，浪费处理机，难于解决复杂同步问题

三. 信号量机制Semaphore

1. 整型信号量：一个用于标志资源数目的整形量S
   1) 除初始化外，仅能通过两个标准的原子操作wait(S)和 signal(S) 来访问
   2) 等待和信号量操作又称为 **P、V** 操作（荷兰文的通过和释放）

3) wait(S){
while (S<=0);
/*死循到S为正*/
S--; }
4) signal(S){
S++;}
5) 原子操作不可中断，因而保证了只有一个进程可修改信号量
6) 未遵循让权等待，会“忙等”
7) S<0时，**绝对值为阻塞进程数**

2. 记录型信号量：除了整型量记录资源数以外，还有链表指针记录阻塞队列
1) typedef struct s{
int value;
struct process_control_block*list;
}semaphore;
2) wait(semaphore*S){
S->value--;
if(S->value<0)
block(S->list);}
3) signal(semasphore*S){
S->value++;
if(S->value<=0)
wakeup(S->list);}
4) value<0，表示该类资源已分配完毕，进程应调用 block 原语，进行自我阻塞，放弃处理机，并插入到阻塞队列list中，做到让权等待
5) value<0时，其绝对值为阻塞队列中进程数
(1) 因而signal后若仍<=0时应调用wakeup原语
6) **互斥信号量：value初值为1**的信号量，其功能为实现互斥访问资源

3. AND型信号量：相当于同时对多个信号量做wait操作，用逻辑AND连接
1) 运行过程中需要的所有资源一次性全分配给进程，待使用完后一起释放
2) 只要尚有一个资源未能分配给进程，其它资源都不分配
3) Swait(S1,S2,…,Sn) {
while(TRUE){
if(S1>=1&&.....&&Sn>=1){
for(i=1;i<=n;i++) Si--;
break;
}else{
place the process in the waiting queue associated with the first Si found with Si<1，and set the program count of this process to the beginning of Swait operation}}}
Ssignal(S1,S2,…,Sn){
while(TRUE){
for(i=1;i<=n;i++){
Si++;
Remove all the process waiting in the queue associated with Si into the ready queue. }}}

4. 信号量集：先测试资源数是否大于下限值t，再减去需求值d个信号量
1) Swait(s1,t1,d1,…,Sn,tn,dn) {
while(TRUE){
if(S1>=t1&&.....&&Sn>=tn){
for(i=1;i<=n;i++) Si-=di;
break;
}else{
place the process in the waiting queue associated with the first Si found with Si<1，and set the program count of this process to the beginning of Swait operation}}}

```
Ssignal(S1,d1,...,Sn,dn){
    while(TRUE){
        for(i=1;i<=n;i++){
            Si+=di;
            Remove all the process waiting in the queue associated with Si into the ready queue. }}}
```

2) Swait(S,d,d)只有一个信号量S，允许它每次申请d个资源
3) Swait(S,1,1)蜕化为一般的记录型信号量(S>1时)或互斥信号量(S=1时)
4) Swait(S,1,0)相当于一个可控开关，只起检查作用。当S≥1时，允许多个进程进入某特定区；当S变为0后，将阻止任何进程进入特定区

四. 信号量的应用

1. 信号量实现进程互斥：为临界资源设置一互斥信号量mutex，初值为1，将各进程访问该资源的临界区CS置于wait(mutex)后signal(mutex)前
    1) mutex取1、0、-1对应都未进入临界区、一个进入、一个进一个阻
    2) wait和signal必须成对出现，wait实现互斥、阻塞，signal实现释放临界资源、唤醒被阻进程
2. 信号量实现前趋关系： P1和P2共享一个公用信号量S，初值为0，在S1后signal(S)；在S2前wait(S)
3. 信号量实现进程同步：s1=0,s2=0;

    写 signal(s1) wait(s2) 循环
    wait(s1) 读 signal(s2) 循环
    或s2初值取1，让即可让写进程和读进程格式一样

五. 管程机制

1. 管程monitor：由共享资源的数据结构，以及由对该数据结构实施操作的一组过程所组成的资源管理模块
    1) 管程的组成：管程名、数据结构说明、对该数据结构进行操作的过程、对共享数据设初值的语句
    2) 封装于管程内的数据仅能被封装于管程内的过程访问，反之亦然
    3) 一次只准许一个进程进入管程，实现了互斥
    4) 特性：
        (1) 模块化，它是基本程序单位，可单独编译
        (2) 抽象：既有数据又有操作
        (3) 信息掩蔽：数据结构和过程实现对外不可见
    5) 管程和进程：
        (1) 进程中的数据结构是私有PCB；管程的是公有数据结构，如数据队列
        (2) 进程由顺序执行有关操作；管程只进行同步和初始化
        (3) 进程实现并发性；管程解决互斥问题
        (4) 进程可像调用子程序一样调用管程；管程只能被动工作
        (5) 进程能并发；管程不能与调用者并发
        (6) 进程有动态性，由创建/撤销而诞生/消亡；管程只是系统中一个供调用的资源管理模块
2. 条件变量
    1) 当一个进程调用了管程，在管程中时被阻塞或挂起，但不释放管程， 则其它进程无法进入管程，被迫长时间地等待。为了解决这个问题，引入了条件变量condition
    2) 在管程中对多个阻塞/挂起原因设置了多个条件变量，对这些条件变量的访问，只能在管程中进行
    3) 条件变量也是一种抽象数据类型，对条件变量的操作仅仅是wait和signal，每个条件变量保存了一个链表，用于记录因该条件变量而阻塞的所有进程
        (1) x.wait：正在调用管程的进程因x条件需要被阻塞或挂起，则调用x.wait将自己插 入到x条件的等待队列上，并释放管程，直到x条件变化。此时其它进程可以使用该管程。
        (2) x.signal：正在调用管程的进程发现x条件发生了变化，则调用x.signal，重新启动 一个因x条件而阻塞或挂起的进程，如果没有，则继续执行原进程，而不产生任何结果
            i. 信号量机制中的signal操作总是要执行s:=s+1操作，因而总会改变信号量的状态

    P唤醒Q后，确定哪个执行，哪个等待，可采用下述两种方式之一进行处理：P等待，直至Q离开管程或等待另一条件/Q等待，直至P离开管程或等待另一条件

Hoare采用了第一种处理方式，而Hansan选择了两者的折衷，他规定管程中的过程所执行的signal操作是过程体的最后一个操作，于是， 进程P执行signal操作后立即退出管程，因而进程Q马上被恢复执行

◆

- 经典进程的同步问题

一. 生产者-消费者问题(The proceducer-consumer problem)

1. 记录型信号量

1) mutex实现互斥进入缓冲池，empty、full实现判断能否继续生产/消费

```
Semaphore mutex=1,empty=n,full=0;
int in=0,out=0;
item buffer[n];
void producer(){
    while(TRUE){
        /*produce the item :nextp*/
        wait(empty);//等到不空时
            wait(mutex);//等到可以进入池时
                buffer[in]=nextp;
                in=(in+1)%n;
            signal(mutex);
        signal(full);}}
void consumer(){
    while(TRUE){
        wait(full);
            wait(mutex);
                nextc=buffer[out];
                out=(out+1)%n;
            signal(mutex);
        signal(empty);
        /*consume the item :nextc*/}}
void main(){
    cobegin
        producer();  consumer();
    coend }
```

2) 两个信号量的wait写反了可能引起死锁

2. AND型信号量：将相邻两个wait/signal合并成一个Swait/Ssignal即可
3. 管程：为缓冲池建立管程，包含put和get过程实现放/取产品；将空/满信号量作为两个条件变量，对应cwait和csignal过程，和一个阻塞队列

二. 哲学家进餐问题

1. 记录型信号量

```
semaphore chopstick[5]={1,1,1,1,1};
while(TRUE){
    wait(chopstick[i]);
    wait(chopstick[(i+1)%5]);
        /*eat*/
    signal(chopstick[i]);
    signal(chopstick[(i+1)%5]);
        /*think*/}
```

1) 可能导致死锁，解决方法：
   (1) 至多允许同时四人拿筷子
   (2) 规定部分人先拿左，部分人先拿右
   (3) 同时检查左右可用，才可以拿，类似下一条

2. AND信号量：两个信号量一起Swait/Ssignal即可，不会死锁

三. 读者-写者问题

1. 记录型信号量

1) 写需要互斥信号量，读可以多进程同时读

2) 无读者在读时才可能在被写，才需要等待写操作/通知写操作
3) ↑读者数应是一个临界资源，需要为其再设置一个互斥信号量
4) 读者数是整型变量，不断++，非零时写者需要等

```
semaphore wmutex=1,rmutex=1;
int readercount=0;
void reader(){
    while(TRUE){
        wait(rmutex);
            if(readercount==0)
                wait(wmutex);
            readercount++;
        signal(rmutex);
        /*read*/
        wait(rmutex);
            readercount--;
            if(readercount==0)
                signal(wmutex);
        signal(rmutex);}}
void writer(){
    while(TRUE){
        wait(wmutex);
            /*write*/
        signal(wmutex);}}
```

2. 信号量集
   1) 可限制读者数最多RN，通过一个初值为RN的信号量不断wait(L,1,1)
   2) Swait(wmutex,1,0)起开关作用，只是在读之前检查是不是在写
   3) 读者数是信号量，不断--，非初值时写者需要等

```
int RN;
semaphore wmutex=1,rMax=RN;
int readercount=0;
void reader(){
    while(TRUE){
        Swait(rMax,1,1);
            Swait(wmutex,1,0);
            /*read*/
        Ssignal(rMax,1);}}
void writer(){
    while(TRUE){
        Swait(wmutex,1,1 , rMax,RN,0);
            /*write*/
        Ssignal(wmutex,1);}}
```

i.
ii.
iii.
iv.
v.
vi.
vii.
viii.
ix. --------我是底线----------

2进程通信、线程

2018年10月17日 13:54

-
- 进程通信

1. 互斥与同步也可视作低级进程通信
   1) 效率低，一次只能从缓冲区取得一个消息
   2) 对用户不透明：OS只提供共享存储器，共享数据结构的设置、数据的传送、进程的互斥与同步必须由程序员实现
2. OS提供的高级通信工具
   1) 使用方便，对用户透明：由OS提供一组隐藏具体细节的通信命令/原语，减少了编程复杂性
   2) 高效：高级通信命令/原语可高效传送大量数据

一. 进程通信的类型

1. 共享存储器系统(Shared-Memory System)：通过某些数据结构或共享存储区进行通信
   1) 基于共享数据结构的通信方式
      (1) 操作系统提供共享存储器，程序员负责设置共用数据结构和处理进程间同步
      (2) 仅适于少量数据，效率低，是低级通信
   2) 基于共享存储区的通信方式
      (1) 内存中划出一块共享区，由进程负责访问控制和数据形式、位置
      (2) 通信前，进程先向系统申请共享存储区中的一个分区，得到关键字、描述符，附加到自己地址空间，读写完成后归还共享存储区
      (3) 由进程控制访问和数据形式、位置。是高级通信
2. 管道pipe通信系统
   1) 管道：用于连接读进程和写进程并实现通信的一个共享文件
   2) 以字符流形式，有效传递大量数据，首创于UNIX
   3) 需要这三方面协调能力：
      (1) 互斥：不能同时读写
      (2) 同步：管道有大量数据后，写进程开始睡眠等待，唤醒读进程；同理管道空以后
      (3) 确定对方是否存在：两者都确认存在时才能通信
3. 消息传递系统(Message passing system)
   1) 以格式化的消息(message)为单位交换数据
   2) 数据全部封装在信息中，利用OS提供的通信命令/原语进行传递
   3) **透明化：通信实现细节被隐藏**，降低了通信程序设计复杂性和错误率
   4) 微内核操作系统无一例外用它与服务器通信；多处理机系统，分布式系统，计算机网络中主要通信工具，计算机网络中，称 message 为报文
   5) 是一种高级通信方式，有两种实现：
      (1) 直接通信：利用OS提供的发送原语，直接发给目标进程
      (2) 间接通信：通过共享中间实体（邮箱）
4. 客户机-服务器系统(Client-Server system)
      (1) 是网络环境的各种应用领域的主流通信机制
   1) 套接字(Socket)：通信标识类型的数据结构
      (1) 起源于20世纪70年代的BSD UNIX
      (2) 包含了通信目的地地址、通信使用的端口号、通信网络传输层协议、进程所在网络地址、

对客户或服务器程序提供的不同系统调用/API函数。是进程通信和网络通信的基本构件

i. 基于文件型：同一机器中，套接字关联到一个本地文件系统，类似管道

ii. 基于网络型：非对称方式（发送者提供接受者命名）不同主机的网络环境下，一对套接字分别属于接受进程/服务器端，一个属于发送进程/客户端。一般客户端发出连接请求后，随机申请一个套接字，主机分配一个专用端口绑定该套接字。服务器端拥有全局公认的套接字和指定接口（如ftp服务器监听端口为21，Web或http的为80）服务器一旦从监听接口收到请求，就立刻接受连接，并发送数据，通信结束后，系统关闭服务器端的套接字套接字，撤销连接

(3) 优势是适用于网络环境，而且每个套接字有唯一的套接字标识符，保证了逻辑链路的唯一性，便于并发传输，采用统一接口，隐藏了通信设施和实现细节

2) 远程过程/函数/方法调用

(1) 调用通信协议(Remote Procedure Call)，使运行于本地主机系统上的进程调用另一台远程主机系统上的进程。程序员只需调用该过程，无需为调用过程编程

(2) 负责处理该过程的进程有两个，是本地客户进程和远程服务器进程，他们也称为网络守护进程，主要负责网络间消息传递，一般都处于阻塞状态，等待消息

(3) 为了使该过程调用透明，引入了存根stub的概念，本地每个可独立运行的远程过程都拥有一个客户存根，本地进程调用远程过程实际上是调用该过程关联的存根；服务器端每个进程也存在服务器存根，这些存根一般也处于阻塞状态，等待消息

(4) 调用远程过程的主要步骤是：本地过程调用者调用远程过程在本地关联的客户存根，传递参数，控制权转给客户存根、客户存根建立包括过程名和调用参数等信息的消息，控制权转给本地客户进程、本地客户进程将消息发送给远程服务器进程、远程服务器将消息转给该消息中的远程过程名对应的存根、服务器存根由阻塞转为执行态，拆开消息，取出参数，调用服务器上关联的过程、服务器端远程过程执行完毕后讲结果返回给关联的服务器存根、服务器存根活动控制权，将结果打包成消息，控制权转移给远程服务器进程、远程服务器进程将消息发送回客户端、本地客户进程接收到消息后将消息存入消息中的过程名对应的客户存根，并将控制权转移给客户存根、客户存放取出结果，返回给本地调用者，并转移控制权。

(5) 即：客户过程的本地调用转化为客户存根，再转化为服务器过程的本地调用，客户和服务器不可见中间步骤

二. 消息传递通信的实现方式

1. 直接消息传递系统

(1) 直接通信方式：发送进程利用OS所提供的发送命令/原语，直接把消息发送给目标进程

1) 直接通信原语

(1) 对称寻址方式：显示提供对方标识符

i. send(receiver,message)；receive(sender,message);

ii. 一旦改变进程的名称，需要检查所有进程，不利于进程模块化

(2) 非对称寻址方式：接收原语中只填写源进程的参数，即通信结束后的返回值，发送不变

i. send(receiver,message)；receive(id,message);

ii. id变量可以设置为发送进程的id或名字

2) 消息的格式

(1) 定长消息：适用于单机系统，减少处理和存储消息的开销，也可用于办公系统的便笺通信

(2) 变长消息：可变长度的消息方便了用户，但可能增加了开销

3) 进程的同步方式

(1) 完成消息发送/接收后有这三种状态

i. 发送接收进程都阻塞：实现进程间紧密同步，无缓冲

ii. 只有接收进程阻塞：为了尽快发更多消息给更多目标，应用最广

iii. 都不阻塞：消息发送/接收意外时间各忙各的，也较常见

4) 通信链路（链路可连接多个结点）

(1) 显式/隐式建立

i. 通信前显式调用建立连接命令/原语，请求系统建立，使用完后拆除，适用于网络

ii. 利用系统的发送命令/原语后系统自动隐式建立链路，主要用于单机系统

(2) 通信方式

i. 单向通信链路：只允许一方发，一方收

ii. 双向通信链路：互相收发

2. 信箱通信

(1) 间接通信方式：通过某种中间实体（如共享数据结构）完成，称该实体为邮箱/信箱，有唯

一标识符。信箱建立在随机存储器的共用缓冲区上，暂存消息。信箱只允许核准用户读取

1) 信箱的数据结构
   (1) 信箱头：存放信箱标识符、拥有者、口令、空格数
   (2) 新箱体：若干存放消息或消息头的信箱格，其数目及大小在创建信箱时确定
2) 信箱通信原语
   (1) 创建和撤销，应有创建者给出名字，属性（公用、私用、共享），共享邮箱需给出共享者
   (2) 发送和接收：Send(mailbox,message); Receive(mailbox,message);
3) 信箱属性
   (1) 私用：用户进程自己建立，是该进程的一部分，会随进程结束而消失。其他用户只能发消息，即单向链路
   (2) 公用：由系统创建，在系统运行期间始终存在。所有系统核准的进程都可用它收发消息，即双向链路
   (3) 共享：进程创建，指明给哪些进程名共用，拥有者和共享者都可收发
4) 四种收发关系
   (1) 一对一：专用通信链路，不熟其他进程干扰
   (2) 多对一：服务进程与多用户进程交互，即客户/服务器交互(client/server interaction)
   (3) 一对多：可用广播方式向多个接收者发消息
   (4) 多对多：用公用邮箱互相收发

三. 直接消息传递系统实例

1) 消息缓冲队列通信机制首先由美国的 Hansan 提出，并在 RC 4000 系统上实现

1. 消息缓冲队列通信机制的数据结构

1) 消息缓冲区

```
typedef struct m{
        int sender; //发送者进程标识符
        int size; //消息长度
        char*text;  //消息正文
        struct message_buffer*next; //指向下一个消息缓冲区的指针
        }message_buffer;
```

2) PCB中有关通信的新增数据项

```
struct message_buffer*mq; //消息队列首指针
semaphore mutex; //消息队列互斥信号量
semaphore sm; //消息队列资源信号量
```

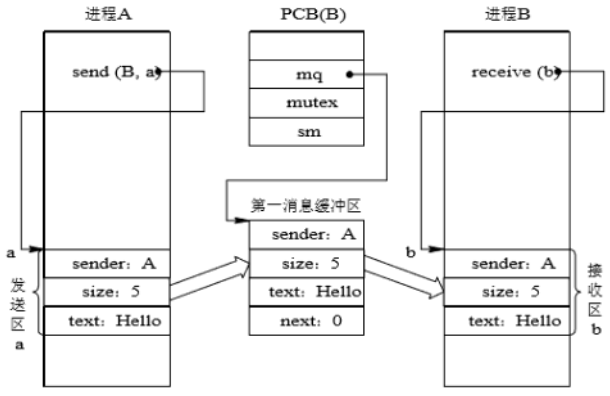


图 2-14 消息缓冲通信

3) 发送原语

1) 利用发送原语发送消息前，应先在内存空间设置发送区a，把消息填入，再调用发送原语，发送原语根据消息长度申请缓冲区i，把a的内容复制到i，再活动接受进程标识符j，把i挂在j的消息队列指针，该队列是临界资源
2) void send(receiver,a){ //向receiver标识符对应进程发送a区的消息

```
getbuf(a.size,i); //申请大小\a.size的名为i的缓冲区
i.sender=a.sender;
i.size=a.size;
strcpy(i.text,a.text);
i.next=0; //用a区的内容初始化i区的内容
getid(PCBset,receiver.j); //找到接收进程的内部标识符
wait(j.mutex);
        insert(j.mq,i); //插到消息队列上
```

signal(j.mutex);
signal(j.sm);}

4) 接收原语
   1) 把消息队列上第一个消息缓冲区i的内容复制到消息接收区b内
   2) void receive(b){
j=internal name; //接收进程内部标识符
wait(j.sm);
wait(j.mutex);
remove(j.mq,i); //取出第一个消息
signal(j.mutex);
ibsender=i.sender;
b.size=i.size;
strcpy(b.text,i.text); //将i的信息复制到b
releasebuf(i);} //释放缓冲区i

-
- 线程的基本概念

1. 20 世纪 60 年代提出进程的概念后，一直都以它作为拥有资源和独立运行的基本单位。直到80 年代中期，才提出了更小独立运行的基本单位——线程(Threads)。 90 年代后，多处理机系统得到迅速发展，线程能更好提高并行执行程度，充分发挥多处理机的优越性，因而在近几年所推出的多处理机 OS 中都引入了线程

一. 线程的引入

1) 线程减少了程序在并发执行时所付出的时空开销

1. 进程的两个基本属性
   1) 可拥有资源的独立单位
   2) 可独立调度和分派的基本单位
2. 程序并发执行所需的时空开销
   1) 创建进程：分配必需的、除处理机以外的所有资源，如内存空间、I/O 设备，建立相应的 PCB
   2) 撤消进程：资源回收，然后再撤消 PCB
   3) 进程切换：保留当前进程的 CPU 环境，设置新选中进程的 CPU 环境， 花费不少的处理机时间
3. 线程作为调度和分派的基本单位
   1) 希望将进程的两个基本属性分开，由OS分开处理，对拥有资源的基本单位，不施以频繁的切换，因而提出了线程
   2) VLSI技术和计算机体系结构的发展出现了对称多处理机SMP系统，提供了良好的硬件基础
   3) 线程作为调度和分派的基本单位，有效地改善了多处理机系统的性能

二. 线程和进程的比较

1) 通常一个进程都拥有若干个线程，至少也有一个线程

1. 调度的基本单位：线程作为调度和分派的基本单位，而进程作为资源拥有的基本单位；线程切换仅需保存和设置少量寄存器内容，切换代价远低于进程
2. 并发性：所有线程都和进程一样能并发，有效提高资源利用率和系统吞吐量。如文字图形、读入数据、拼写检查放在不同线程；如一个线程专门用于监听客户请求，及时创建其他线程来处理请求
3. 无系统资源：只有TCB、程序计数器、用于保存局部变量、状态参数、和返回地址的寄存器和堆栈
4. 无独立性：每个进程都拥有独立地址空间和其他资源，除了共享全局变量外，不允许其他进程访问；而线程往往是共享进程的内存地址空间和资源
5. 系统开销小：进程创建和撤销除了PCB以外还有各种资源要分配和回收、切换上下文也比线程慢、同步和通信也没线程快，某些OS里线程的切换、同步和通信都无需内核干预
6. 支持多处理机系统：传统进程只能运行在一个处理机上；有多线程的进程能在多个处理机上并行

三. 线程的状态和控制块

1. 三个运行状态：执行：已获得处理机；就绪：具备CPU以外执行条件；阻塞：执行时因某事件而暂停
2. 线程控制块TCB：线程唯一标识符；寄存器：程序计数器PC、状态寄存器、通用寄存器的内容；运行状态；优先级；存储区：切换时的现场信息、线程相关统计信息；信号屏蔽；堆栈指针：用户态和核心态的两套函数调用的局部变量、返回地址的堆栈的指针；

3. 多线程OS中的进程
    1) 拥有资源的基本单位：用户地址空间、同步通信机制的空间、文件、设备、地址映射表
    2) 并发性：线程都属于进程，线程可并发执行
    3) 不是可执行实体：进程执行是指进程某些线程在执行。进程的挂起等状态操作是对所有线程进行

- ◆
- ◆ 线程的实现

单进程和单线程系统（DOS）
多进程和单线程系统（UNIX）
单进程和多线程系统（Java Run-time System）
多进程和多线程系统（Windows2000、Solaris、Mach）

一. 线程的实现方式

1. 内核支持线程(Kernel Supported Threads)
    1) KST与进程一样与内核紧密相关，TCB存储在内核空间，便于内核感知和控制它
    2) 调度以线程为单位，轮转调度算法中线程多的进程可获得更多运行时间
    3) 优点：多处理机中并行、不怕阻塞、切换快开销小、内核本身也多线程
    4) 缺点：要在用户态到核心态不断切换，因为线程调度和管理靠内核实现
2. 用户级线程(User Level Threads)
    1) 仅存在于用户空间，无需内核支持，TCB也在用户空间，内核无法感知
    2) 调度以进程为单位，轮转调度算法中多线程进程中的线程只能获得很少运行时间
    3) 优点：全程用户态，不用切换、调度算法可由进程自己选择，与OS低级调度算法无关、管理方式是程序代码一部分，无线程平台也可以实现
    4) 缺点：进程阻塞使所有线程阻塞、只能单CPU
3. 组合方式
    1) 通过时分多路复用KST实现：KST对应多个ULT。程序员可自由调整KST数目
    2) 三种对应连接方式：
        (1) 多对一模型：多个ULT，仅当需要访问内核时，映射到一个KST，每次只允许射一个
            i. 管理开销小，效率高；但KST阻塞了就会阻塞整个进程，不支持多处理器
        (2) 一对一模型：每个ULT对应一个KST
            i. 不怕阻塞，更好的并发，支持多处理器；但开销极大，KST数是需要限制的
        (3) 多对多模型：许多ULT映射到同样数量或更少数量的KST
            i. 可并行，可多处理器，开销少，效率高，不怕阻塞

二. 线程的实现

1. 无论线程和进程都直接间接需要内核支持
2. 内核支持线程的实现
    1) 创建一个新进程时，便为它在内核中分配一个任务数据区 PTDA(Per Task Data Area)，存储若干 TCB
    2) 只要线程数目未超过系统允许值(通常为数百个)，系统可再为之配新的 TCB 空间；在撤消一个线程时，有时为了节省开销，不收回资源和TCB，方便新进程直接利用
3. 用户级线程的实现
    1) 运行时系统(Runtime System)：管理和控制线程的函数/过程的集合
        (1) 切换时不用转入核心态，而是由运行时系统的线程切换函数将CPU状态保存在该现场的堆栈，调度其他线程，将新线程的CPU状态装入对应CPU寄存器，再切换栈指针和程序计数器，开始新程序运行，这种切换无需进入内核，因而很快
        (2) 资源总是内核管理的。进程通过系统调用，它通过软中断机制（如trap）进入内核，由内核完成资源分配；运行时系统则负责接收线程的资源请求，再通过相关系统调用分配资源
    2) 内核控制线程/轻型进程 LWP(Light Weight Process)
        (1) 每个LWP都有自己的数据结构，可以共享进程的资源，通过系统调用获得内核提供的服务
        (2) LWP相当于缓冲池，又称线程池
        (3) 只有连接到LWP上的线程才能与内核通信，且内核只能看到LWP（连接方式见一.3）
        (4) 内核级线程阻塞后，与之相连的多个LWP也将随之阻塞，与LWP相连的多个用户级线程也阻塞，不过其他LWP仍能执行。LWP阻塞后，线程不能访问内核，但其他操作仍能执行。

类似于进程执行系统调用时，进程会阻塞

(5)

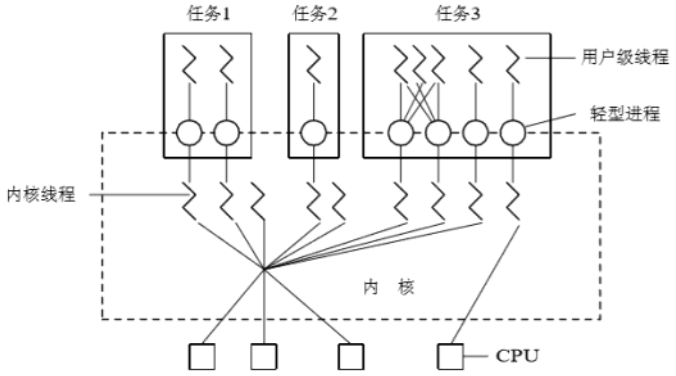


图 2-16 利用轻型进程作为中间系统

三. 线程的创建和终止

1) 线程也由创建而产生，由调度而执行，由终止而消亡

1. 线程的创建
   1) 通常进程刚启动时仅有一个线程，用于创建其他进程
   2) 调用线程创建函数/系统调用，提供参数：程序入口指针、堆栈大小、调度优先级
   3) 创建完返回线程标识符
2. 线程的终止
   1) 调用相应的函数/系统调用来终止
   2) 系统线程等线程建立后就会一直运行下去而不被终止
   3) 多数OS的线程终止后不立即释放资源，仅当进程中的其他线程执行了分离函数后，终止的线程才与资源分离，其资源才能被其他线程利用。释放前，调用者线程课调用“等待线程终止”的命令来连接该线程，然后（阻塞等待）到该线程终止时，调用者继续执行

i.

ii.

iii.

iv.

v.

vi.

vii.

viii. --------我是底线----------

3处理机调度

2018年10月22日 9:00

-
- 处理机调度的层次和调度算法的目标

1. 系统吞吐量、资源利用率、作业周转时间、响应及时性很大程度上取决于处理机调度性能好坏
2. 调度的实质是一种资源分配

一. 处理机调度的层次

1. 高级调度(High Level Scheduling)
   1) 作业调度：调度对象是作业，决定将哪些作业**从外存**的后备队列调入内存，并创建进程、分配资源、放入就绪队列
   2) 长程调度(LongTerm Scheduling)：周期长，约几分钟，一批作业完成才重新调，运行频率低因而算法可以花较多时间
   3) 出现在多道批处理系统
2. 低级调度(Low Level Scheduling)
   1) 进程调度：调度对象是进程或内核级线程，决定哪个进程分配到**处理机**
   2) 短程调度(ShortTerm Scheduling)：运行频率高，耗时极短，算法不复杂
   3) 是最基本的调度，多道批处理、分时、实时系统都须配置这级调度
3. 中级调度(Intermediate Level Scheduling)
   1) 内存调度：为提高内存利用率和系统吞吐量而挂起进程/改为就绪驻外存状态，再由中级调度调入内存
   2) 中程调度(Medium-Term Scheduling)：以上两者之间
   3) 本质上就是**存储管理器对换**功能

二. 处理机调度算法的目标

1. 各系统共同的目标：
   1) 资源利用率：让处理机等资源尽可能的保持忙碌状态
      (1) cpu利用率=CPU有效工作时间/
      (CPU有效工作时间+CPU空闲等待时间)
   2) 公平性：诸进程都应获得合理的CPU时间，不发生进程饥饿现象
      (1) 相对：同类进程同服务，不同紧急程度或重要性提供不同服务
   3) 平衡性：尽可能使各种CPU和外部设备处于忙碌状态
   4) 策略强制执行：包括安全策略，需要时，即使会延迟某些工作，也要对所有制订的策略都准确执行
2. 批处理系统特有的目标
   1) 平均周转时间短：尽可能提高系统资源利用率，使大多数用户都满意
      (1) 作业周转时间：作业被提交给系统开始，完成为止的时间间隔
      (2) 包括：外存后备队列上等、就绪队列上等，CPU上执行、阻塞时
      (3) ↑只有第一项只会发生一次
      (4) 平均周转时间&**平均带权周转时间**（Ti是周转时间，Ts是系统提供

（处理机）服务时间）

$T=\frac{1}{n}\left[\sum_{i=1}^{n}T_i\right]$	$W=\frac{1}{n}\left[\sum_{i=1}^{n}\frac{T_i}{T_s}\right]$

2) 系统吞吐量高：尽可能单位时间内完成更多作业
   (1) 尽可能选择短作业
3) 处理机利用率高，这是衡量系统性能十分重要的指标
   (1) 尽可能选择计算量大的作业

3. 分时系统特有的目标
   1) 响应时间快：分时系统调度算法重要准则
      (1) **响应时间**：从键盘提出请求到屏幕上显示出处理结果为止的时间
      (2) 包括：键入信息并传送进处理机、处理机处理信息、响应信息回送到终端显示器
   2) 均衡性：响应时间快慢与服务的复杂性相适应
4. 实时系统特有的目标
   1) 截止时间保证：尤其是HRT任务必须保证，否则会有难以预料的后果，SRT也基本要保证
      (1) 截止时间：某任务开始执行/完成执行的最迟时间
   2) 可预测性：为了提高实时性，需要预测
      (1) 如多媒体系统最好用双缓冲，提前处理下一帧

三. *调度队列模型*

1. 仅有进程调度的调度队列模型
   1) 分时系统，通常仅设置进程调度，用户键入的命令和数据都直接送入内存。对于命令，由 OS 为之建立一个进程
   2) 系统可根据调度算法把处于就绪状态的进程组织成栈、树或无序链表
      (1) 如，分时系统中，常把就绪进程组织成 FIFO 队列形式，新进程被挂在就绪队列的末尾，按时间片轮转方式运行
   3) 每个进程在执行时都可能出现以下三种情况
      (1) 任务在给定时间片内完成，进程便在释放处理机后进入完成状态
      (2) 任务在本次时间片内未完成，OS 便将该任务再放入就绪队列末尾
      (3) 进程被阻塞，被 OS 放入阻塞队列
   4)

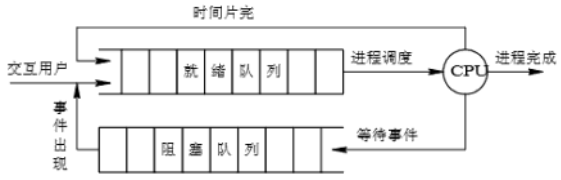


图 3-1 仅具有进程调度的调度队列模型

2. 具有高级和低级调度的调度队列模型
   1) 批处理系统，不仅需要进程调度，而且还需有作业调度。先由作业调度从外存的后备队列中选择一批作业调入内存，并为它们建立进程，送入就绪队列，再由进程调度按照一定的进程调度算法选择一个进程，把处理机分配给该进程
   2) 批处理系统最常用的是最高优先权优先调度算法，其就绪队列的形式：
      (1) 优先权队列：新进程根据优先权高低，插入相应位置，调度程序总把处理机分配给队首进程

(2) 无序链表：新进程总挂在链尾，每次调度时，依次比较该链中各进程的优先权，选出最高优先权。显然，效率较低

3) 设置多个阻塞队列：大、中型系统通常都按阻塞事件设置若干阻塞队列。系统较大时，阻塞队列中的进程数可以达到数百个，这将严重影响对阻塞队列操作的效率

4)

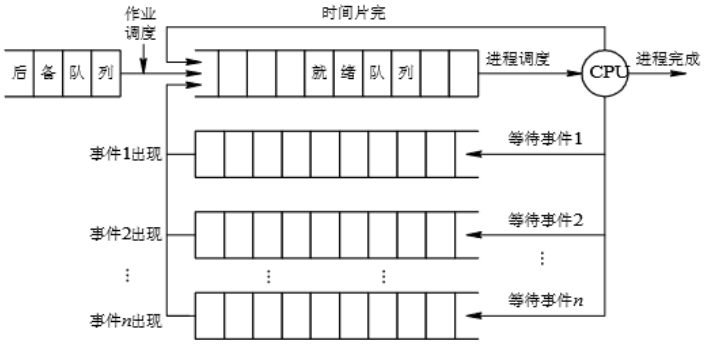


图 3-2 具有高、低两级调度的调度队列模型

3. 同时具有三级调度的调度队列模型

1) 当在 OS 中引入中级调度后，人们可把进程的就绪状态分为内存就绪(表示进程在内存中就绪)和外存就绪(进程在外存中就绪)。类似地，也可把阻塞状态进一步分成内存阻塞和外存阻塞两种状态。在调出操作的作用下，可使进程状态由内存就绪转为外存就绪，由内存阻塞转为外存阻塞；在中级调度作用下，又可使外存就绪转为内存就绪

2)

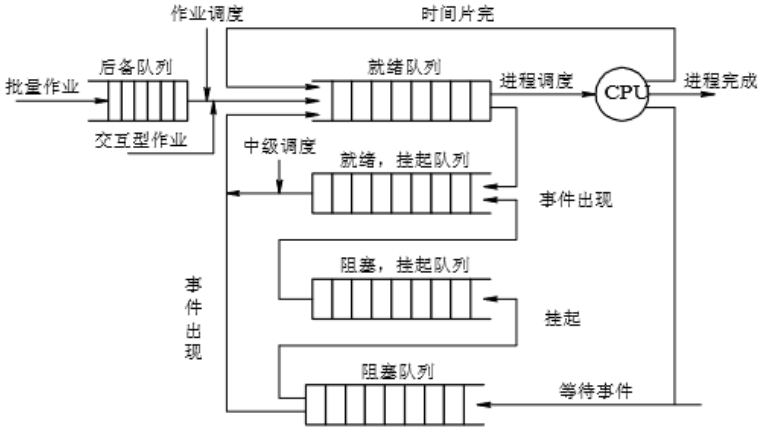


图 3-3 具有三级调度时的调度队列模型

i.

ii.

iii.

iv.

v.

vi.

vii. --------我是底线----------

3调度算法

2018年10月22日 9:00

-
- 作业和作业调度

1. 多道批处理系统中，作业是用户提交给系统的一项相对独立的工作

一. 批处理系统中的作业

1. 作业和作业步
   1) 作业(Job)除了包含通常的程序和数据， 还应配有一份作业说明书，系统根据说明书控制程序的运行
      (1) 批处理系统中，以作业为基本单位从外存调入内存
   2) 作业步(Job Step)：作业的每一个加工步骤
      (1) 往往把一个作业步的输出作为下一个作业步的输入。 例如，一个典型的作业可分成三个作业步：编译、连接装配、运行
2. 作业控制块(Job Control Block)
   1) 是作业在系统中存在的标志
   2) 包含：标识符、用户名、用户帐户、作业类型(CPU 繁忙型、I/O 繁忙型、批量型、终端型)、作业状态、调度信息(优先级、作业已运行时间)、资源需求(预计运行时间、 要求内存大小、要求 I/O 设备类型和数量等)、进入系统时间、开始处理时间、作业完成时间、作业退出时间、资源使用情况等
   3) 由系统的“作业注册”程序建立JCB，排到后备队列；由系统的调度程序按算法调度它们进入内存；执行时系统根据JCB和说明书对作业进行控制；结束后由系统回收资源，撤销JCB
3. 作业运行的三个阶段和三种状态
   1) 收容阶段：作业输入到硬盘，创建JCB，放入后备队列
      (1) 此时是后备状态
   2) 运行阶段：分配资源，建立进程，放入就绪队列
      (1) 第一次就绪到运行结束前都是运行状态
   3) 完成阶段：系统的“终止作业”程序回收JCB和资源，并将运行结果信息形成文件并输出
      (1) 运行结束后进入完成状态

二. 作业调度的主要任务

1) 作业调度的主要功能是根据JCB，审查系统资源，从外存的后备队列中选取某些作业调入内存，并创建进程、分配资源，再插入就绪队列。有时把作业调度称为接纳调度(Admission Scheduling)

1. 接纳多少作业，取决于多道程序度(Degree of Multiprogramming)
   1) 多道程序度：允许多少个作业同时在内存中运行
   2) 太多容易内存不足而中断运行；太少使平均周转时间显著延长
   3) 取决于系统规模、运行速度、作业大小、系统性能
2. 接纳哪些作业，取决于采用的调度算法
   1) 最易：FCFS；最常用：SJF；较常用：PSA；最好：HRRN
   2) 作业进入批处理系统后总是先驻留在外存的后备队列上，因而需要作业调度。分时系统无需作业调度，直接将命令或数据送入内存以实现及时响应，只需用接纳控制措施限制进入系统的用户数

三. FCFS和SJF

1. **先来先服务**调度算法(First-come first-served)
   1) 相当于只考虑等待时间最长的作业
   2) 可组合使用，如先按优先级设多个队列，每个队列里FCFS

2. **短作业优先**(Short job first)
    1) 长短以作业要求的运行时间来衡量，多数是短的
    2) 缺点：
        (1) 运行时间估计短了会提前终止作业
        (2) 不考虑等待时间，长作业周转时间明显增长，出现饥饿现象
        (3) 无法实现人机交互
        (4) 不考虑紧迫度，高紧迫性作业不能保证及时处理

四. PSA和HRRN

1. 优先级调度算法(Priority-scheduling algorithm)，详见进程调度
    1) 外部基于作业紧迫度，赋予作业一个优先级，保证高紧迫性作业先运行
2. **高响应比优先调度**算法(High Response Ratio Next)
    1) 是唯一一个基本只用于作业调度的算法
    2) $$\text{优先权} = \frac{\text{等待时间} + \text{要求服务时间}}{\text{要求服务时间}}$$
    3) $$R_P = \frac{\text{等待时间} + \text{要求服务时间}}{\text{要求服务时间}} = \frac{\text{响应时间}}{\text{要求服务时间}}$$
    4) 以上两个公式本质上是一个，但前者称为优先权，后者称为**响应比**
    5) 实现了FCFS和SJF的折中：等待时间差不多时，类似SJF；要求服务时间差不多时类似FCFS；长作业不会饥饿
    6) 同时考虑了等待时间和运行时间，改善了处理机调度性能，但每次调度前都要计算Rp增加了系统开销
        -
        - 进程调度

一. 进程调度的任务、机制和方式

1. 进程调度的任务：保存处理机现场、按算法选取进程、把处理器分配进程
2. 进程调度机制
    1) 排队器：把转为就绪状态的进程插入对应就绪队列
    2) 分派器：从就绪队列取出应调度的进程，并选新进程的上下文切换
    3) 上下文切换器：
        (1) 保存当前进程的上下文到PCB，再装入分派程序的上下文
        (2) 移出分派程序的上下文，把新进程的CPU现场装入各CPU寄存器
        (3) 需要执行大量load和store等操作指令，执行一次上下文切换大约可执行上千条其他指令
        (4) 或用两套硬件分别供系统态、应用程序使用，则上下文切换只需改变指针指向不同寄存器组
3. 进程调度方式
    1) 非抢占方式(Nonpreemptive Mode)
        (1) 决不会因为时钟中断等原因而抢占正在运行进程的处理机
        (2) 引起调度的原因：执行完毕，或因发生某事件不能继续执行；因提出 I/O 请求而暂停执行；执行了 wait、Block 、Wakeup 等原语

2) 抢占方式(Preemptive Mode)
   (1) 允许调度程序根据以下原则去暂停正在执行的进程
      i. 优先权原则、短作业(进程)优先原则、时间片原则
   (2) 实现了分时系统的人机交互、实时系统的HRT

二. **轮转调度**算法Round Robin

1) 分时系统中基于时间片的轮转使n个就绪进程都约获得1/n的处理机时间

1. 轮转法的基本原理
   1) 按FCFS排就绪队列，隔一定时间(如30ms)产生一次中断，调度队首进程
   2) 老师说默认刚结束的进程出现在队尾，即新到进程可能在倒数第二
2. 进程切换时机：执行中程序提前完成、时间片结束（然后移至队尾）
3. 时间片大小的确定
   1) 太短导致频繁调度、切换，增加系统开销
   2) 太长相当于退化成FCFS，无法满足短作业和交互
   3) 因此时间片一般取略大于一次交互的时间，缩小响应时间

三. **优先级调度**算法Priority-scheduling algorithm

1) 轮转法没有考虑紧迫性，即默认紧迫性相同

1. 调度算法类型
   1) 非抢占式：直至执行完成或自行放弃，都不重新分配处理机
   2) 抢占式：一旦出现新进程，立刻比较优先级，若高于执行中进程，立即重新分配处理器。适用于实时性要求高的系统
2. 优先级的类型
   1) 静态优先级：创建进程时确定，是一个不变的整数
      (1) 确定依据：
         i. 进程类型：如系统进程优先权高(接收、对换、磁盘 I/O 等进程)
         ii. 进程对资源的要求：要求少的，优先级高
         iii. 用户要求：紧迫度及用户所付费用的多少
      (2) 简单易行、系统开销小、不精确、低优先级进程可能饥饿
   2) 动态优先级：先赋初值，再随进程推进/时间增加而改变
      (1) 相同初值时，类似FCFS
      (2) 若随等待时间增加而增加优先级，则短优先级能获得处理机
      (3) 若采用抢占式，还能防止长作业长期垄断处理机

四. 多队列调度算法

1. 将不同类型或性质的进程分配在不同的就绪队列，不同队列采取不同调度算法，不同进程可以设置不同优先级，不同队列也可以有不同优先级
2. 多处理机系统可以为每个处理机设置一个单独的就绪队列
3. 需相互合作的进程或线程也可分配到一组处理机的多个就绪队列，实现并行

五. **多级反馈队列**(multileved feedback queue)调度算法

1) 它不用事先安排各进程所需的执行时间，公认较好的进程调度算法

1. 调度机制
   1) 设置多个就绪队列，优先级逐个降低，时间片逐个增倍

(1)

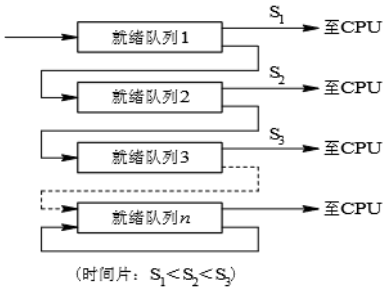


图 3-7 多级反馈队列调度算法

2) 各队列内采用FCFS，新进程放入第一个队列末尾。时间片结束后若未完成，则进入下一队列末尾，到某个队列N后开始轮转
3) 按队列优先级调度，即只有1~(i-1)队列都为空时才调用i队列首进程。若有新进程进入高优先级队列，应立即把当前进程放入原队列末尾

2. 调度算法的性能
   1) 一般规定第1队列时间片略大于交互所需处理时间，满足用户需求
      (1) 终端型用户：交互型小作业，一般第1队列就能完成
      (2) 短批处理作业用户：一般在前三队列都能完成，周转时间仍短
      (3) 长批处理作业用户：第n队列中RR，一般不必担心长期无法处理

六. **基于公平原则**的调度算法

1) 其他算法并不保证作业占用多少处理机时间，不一定公平

1. 保证调度算法：向用户保证性能，n个同类型进程各获得处理机时间1/n，需要这些功能：
   1) 跟踪计算每个进程实际获得的处理时间
   2) 计算应获得的处理时间，即创建以来的时间除以进程数n
   3) 计算执行时间比率，即1）/2）
   4) 比较各进程的比率
   5) 将处理机分配给比率最小的进程，运行到超过第二小的比率
2. 公平分享调度算法
   1) 若各用户所拥有的进程数不同，保证调度算法对用户就不公平
   2) 同一用户有多个进程，则该用户的每个进程活动的时间会减少

- ◆
- ◆ 实时调度

一. 实现实时调度的基本条件

1. 提供必要信息
   1) 就绪时间：成为就绪态的起始时间、周期任务是预知的一串时间序列
   2) 开始截止时间和完成截止时间，一般只需一个
   3) 处理时间：开始执行到完成所需时间
   4) 资源要求：执行时所需的一组资源
   5) 优先级：错过截止时间会引起故障，则有“绝对”优先级，无重大影响的就赋予“相对”优先级，供调度程序参考。（即硬/软实时任务）
2. 系统处理能力强
   1) m 个周期性的硬实时任务，处理时间为 Ci，周期时间为 Pi， 在单处理

机情况下，必须满足下面的限制条件，系统才可调度

(1) $$\sum_{i=1}^{m}\frac{C_i}{P_i}\leqslant 1$$

2) 若是N个处理机的多处理机系统，限制条件改为

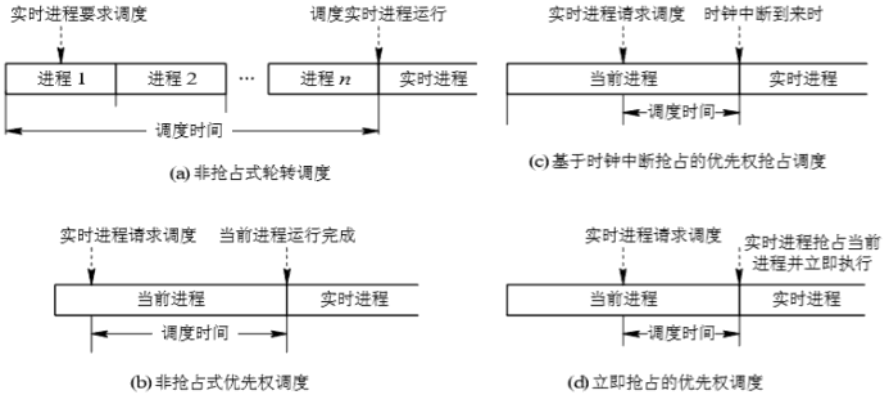
(1) $$\sum_{i=1}^{m}\frac{C_i}{P_i}\leqslant N$$

3) 都只是最低要求，需要留有余地给调度算法、任务切换、消息传递等

3. 采用抢占式调度机制：满足HRT任务
   1) 如果已知任务开始截止时间，也可用非抢占机制，执行完关键程序和临界区后，进程及时阻塞自己
4. 具有快速切换机制
   1) 对外部中断的快速响应能力：快速硬件中段机构，保证紧迫任务
   2) 快速的任务分派能力：减少切换任务的时间开销

二. 实时调度算法的分类

1. 非抢占式调度算法
   1) 非抢占轮转：时间片结束后挂在轮转队列末
      (1) 偏软，有数秒至数十秒的响应时间
   2) 非抢占优先：新到的高优先级排在队首
      (1) 偏硬，响应时间数秒至数百毫秒
2. 抢占式调度算法
   1) 基于时钟中断的抢占式优先级：等时钟中断发生时调度高优先级任务
      (1) 偏软，调度延迟几十至几毫秒
   2) 立即抢占优先级(Immediate Preemption)：快速响应外部事件中段，只要当前任务未处于临界区，就能立即剥夺处理器给请求中断的紧迫任务



3.

图 3-8 实时进程调度

三. **最早截止时间优先**(Earliest Deadline First)算法

1) 截止时间越早，优先级越高

1. 非抢占：用于非周期实时任务（假设截止时间3<4<2）

1)

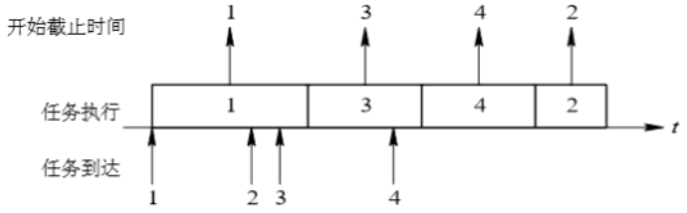


图 3-9 EDF 算法用于非抢占调度的调度方式

2. 抢占：用于周期实时任务（第一行是要求，二三行是任务固定优先级，第四行是正解）

1)

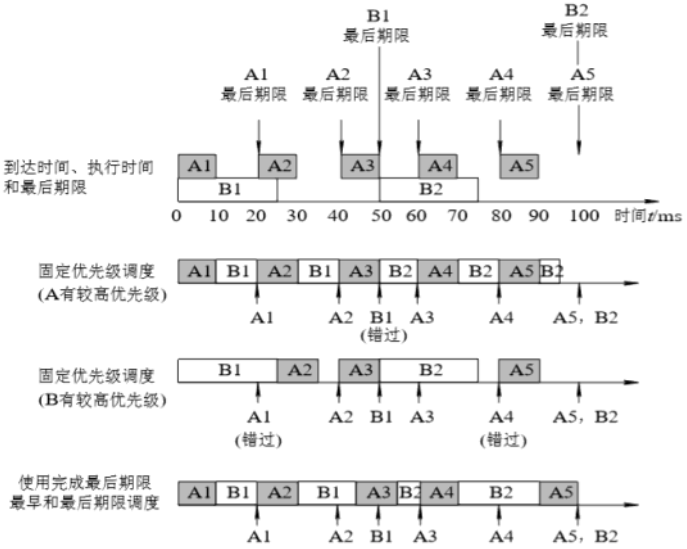


图 3-10 最早截止时间优先算法用于抢占调度方式之例

四. **最低松弛度优先**(Least Laxity First)算法

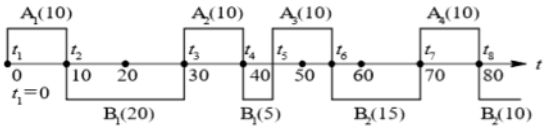
1. 紧急程度=**松弛度=必须完成时间-其本身的运行时间-当前时间**
   1) 相当于距离deadline还剩的时间
2. 主要用于可抢占调度，完成周期性实时任务
3. 老师们认为每当有新进程进入，就应当比较松弛度，但图上是每当有进程松弛度为零才调度
4. 

图 3-12 利用 LLF 算法进行调度的情况

五. 优先级倒置问题(priority inversion problem)

1. 优先级倒置的形成：低优先级进程/线程，延迟或阻塞了高优先级进程/线程。如占用了互斥资源
2. 其解决方法
   1) 进入临界区后不允许处理机被抢占。仍可能导致高优先级进程等待很久
   2) 动态优先级继承：若低优先级进程抢占了高优先级进程想要的资源，则让低优先级进程继承到该阻塞进程的优先级，防止被中优先级插入

i.

ii.

iii.

iv.

v.

vi.

vii. --------我是底线----------

3死锁

2018年10月31日　　13:39

-
- 死锁概述

一. 资源问题

1) 临界资源=要互斥访问的、不可抢占性资源，互斥≈不可共享

1. 可重用性资源和消耗性资源

1) 可重用性资源：可供用户重复使用多次的资源

i. 不可共享：每个单元只能分配给一个进程

ii. 使用顺序：请求资源、使用资源、释放资源

iii. 单元数目相对固定：运行期间不能创建/删除

(1) 请求和释放通常用系统调用。设备：request/release；文件：open/close；互斥：wait/signal

(2) 计算机系统中大多数资源都可重用

2) 可消耗性资源/临时性资源：由进程动态地创建/消耗的资源

i. 单元数目在程序运行期间不断变化

ii. 被进程不断创造而使单元数目增加（放入缓冲区）

iii. 由进程消耗，不返回给该资源类

(1) 如通信消息就是可消耗性资源，由生产进程创建，消费进程消耗

2. 可抢占性资源和不可抢占性资源

1) 可抢占性资源：如低优先级进程的处理机被高优先级进程抢占

(1) 又如内存紧张时被挂到外存，即内存被抢占

(2) 不会引起死锁

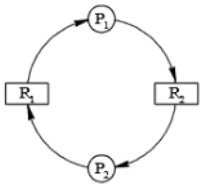
2) 不可抢占性资源：一旦被分配给进程，就只能被进程自行释放

(1) 如光盘刻录机、磁带机、打印机

二. 计算机系统中的死锁deadlock

1. 竞争不可抢占性资源

☑1) 如多进程请求打开多文件，资源分配图形成回路，说明已进入死锁

2) 

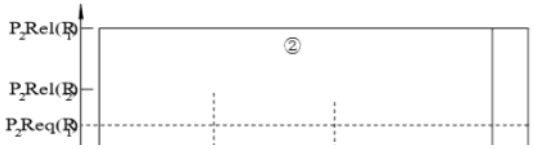
2. 竞争可消耗性资源

1) 如通信时都在等收到上家发送的消息，再向下家发送

3. 进程推进顺序不当

1) 合法：不会引起死锁的推进顺序

2) 非法：如四号路线进入了不安全区d，可能死锁



3)

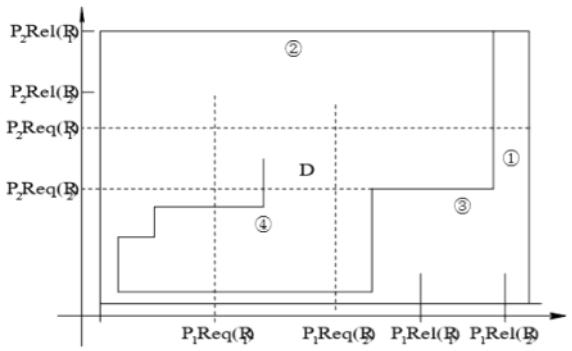


图 3-15　进程推进顺序对死锁的影响

三. 死锁的定义、必要条件、处理方法

1. **死锁：一组进程中各进程都在等待仅能由该组进程中其他进程引发的事件**
2. 产生死锁的**必要**条件：
   1) **互斥**：资源被其他进程占用，请求进程只能等待其释放
   2) **请求和保持**：进程已保持至少一个资源不放，还想请求新资源
   3) **不可抢占**：资源只在被进程使用完后释放
   4) **循环等待**：存在一个 进程-资源-进程-资源…… 的循环链
3. 处理死锁的方法
   (1) 预防：设置某些限制条件去破坏必要条件
   (2) 避免：资源分配过程中防止系统进入不安全状态
   (3) 检测：及时发现死锁的发生
   (4) 解除：发现死锁后立刻撤销部分进程

1) 从上至下，预防程度弱，资源利用率高，阻塞频度低，并发程度高

-
- 预防死锁

一. 破坏“请求和保持”条件

1. 第一种协议：所有程序必须一次性申请完运行过程中可能需要的全部资源
   1) 一次分配完，破坏了请求；（申请不到全部就）不分配资源，破坏了保持
   2) 简单易行又安全，但
      (1) 资源利用率严重恶化：很多资源在运行时只用一会，被严重浪费
      (2) 进程饥饿现象：个别资源被占用使等待进程迟迟不能开始运行
2. 第二种协议：获得某一时期所需资源后便开始运行，边运行边释放资源
   1) 更快地完成任务，提高设备利用率，减少进程饥饿几率

二. 破坏“不可抢占”条件

1. 新资源请求不能满足时，释放已保持的所有资源，即可被其他进程抢占
2. 难实现，代价大，可能是前后两次运行的信息不连续，可能因反复申请释放导致进程执行被无限推迟，延长周转时间，增加系统开销，降低吞吐量

三. 破坏“循环等待”条件

1. “按序分配”：对资源类型做线性排序，赋予唯一序号，输入设备低，输出设备高；请求资源顺序必须按序号递增；想请求低序号类资源前，必须先释放同序号和高序号的资源。占据高序号资源的进程一定能不断推进
2. 资源利用率和系统吞吐量都得到改善，但

1) 资源排序限制了新类型资源的增加
2) 作业使用资源顺序与排序不同时会造成资源浪费
3) 限制了用户简单、自主的编程

-
- 避免死锁

一. 系统安全状态

1. 安全状态：能按某进程推进顺序为每个进程分配资源，使各进程都顺利完成
   1) 安全序列：↑这种推进顺序
   2) 不安全状态：无法找到安全序列的状态
   3) 不安全状态是死锁的必要非充分条件，安全是不死锁的充分非必要条件
2. 安全状态之例
   1)

进　程	最大需求	已 分 配	可　用
P_1	10	5	3
P_2	4	2	
P_3	9	2	

   2) 如图，<P2，P1，P3>即为安全序列，按此顺序能完成每个进程
3. 由安全状态向不安全状态的转换
   1) 不按照安全序列分配资源就可能会转换进不安全状态
   2) 资源分配前必须计算安全性，会进入不安全状态，则不分配

二. 银行家算法

1) Dijkstra为银行系统设计的，防止现金贷款时不能满足所有客户需求

1. 银行家算法中的数据结构
   1) 可利用资源向量Available：含有 m 个元素的数组
      (1) 每一个元素代表一类可利用的资源数目
      (2) 初值是系统中该类全部可用资源的数目，会动态地改变
      (3) Available[j]=K 表示系统中现有 Rj 类资源 K 个
   2) 最大需求矩阵 Max：n×m 的矩阵
      (1) 定义了 n 个进程中的每一个进程对 m 类资源的最大需求
      (2) Max[i][j]=K 表示进程 Pi 需要 Rj 类资源的最大数目为 K
   3) 分配矩阵 Allocation：n×m 的矩阵
      (1) 定义了系统中每一类资源当前已分配给每一进程的资源数
      (2) Allocation[i][j]=K 表示进程 Pi 当前已分得 Rj 类资源 K 个
   4) 需求矩阵 Need：n×m 的矩阵
      (1) 表示每一个进程尚需的各类资源数
      (2) Need[i][j]=K 表示进程 Pi 还需要 Rj类资源 K 个方能完成其任务
      (3) Need[i][j] = Max[i][j] - Allocation[i][j]

2. 安全性算法

1) 设置两个向量
   (1) 系统剩余可分配资源向量Work，有 m 个元素，初值同Available
   (2) 分配结束向量Finish，表示系统是否有足够的资源分配给进程，使之运行完成。初值全为false；有足够资源时，再令Finish[i]:=true
2) 寻找满足下述条件的进程

(1) Finish[i]=false;

(2) Need[i][j]≤Work[j];

3) 找到后分配资源，顺利完成后再全部释放

(1) Work[j] += Allocation[i][j];

(2) Finish[i] = true;

(3) go to step 2;

4) 如果第二步没找到，且存在Finish[i] = true，说明系统处于不安全状态

3．银行家算法

1) 设 Request i：进程 Pi的请求向量

(1) Request i[j]=K 表示进程 Pi 请求 K 个 Rj类型的资源

2) 发出资源请求后按下属步骤操作：

(1) 确认 Request i[j]≤Need[i,j]； 不成立说明出错，所需资源数超过它声明的最大值

(2) 确认 Request i[j]≤Available[j]； 不成立表示尚无足够资源，Pi 须等待

(3) 系统尝试分配资源，并修改下面的数值：

i. Available[j] -= Request i[j];

ii. Allocation[i][j] += Request i[j];

iii. Need[i][j] -= Request i[j];

(4) 执行安全性算法，确认安全才正式分配资源给进程 Pi，以完成本次分配；否则将本次的试探分配作废，恢复原状，让进程 Pi等待

4. 银行家算法之例

1) P1请求资源Request1(1，0，2)，系统按银行家算法进行检查

(1) Request1(1，0，2)≤Need1(1，2，2)

(2) Request1(1，0，2)≤Available1(3，3，2)

(3) 系统先假定可为 P1分配资源，并修改 Available，Allocation1和 Need1向量，由此形成的资源变化情况如图 3-16 中的圆括号所示

资源 / 情况 / 进程	Max			Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0	7	5	3	0	1	0	7	4	3	3	3	2
										(2	3	0)
P_1	3	2	2	2	0	0	1	2	2			
				(3	0	2)	(0	2	0)			
P_2	9	0	2	3	0	2	6	0	0			
P_3	2	2	2	2	1	1	0	1	1			
P_4	4	3	3	0	0	2	4	3	1			

图 3-16 T_0 时刻的资源分配表

(1) T0时刻的安全性：如图 3-17 ，利用安全性算法可知存在着安全序列{P1，P3，P4，P2，P0}，故系统是安全的，可按此序列分配资源（然而这个例子接下来演示的是如何作死，进入不安全序列）

资源 / 情况 / 进程	Work			Need			Allocation			Work+Allocation			Finish
	A	B	C	A	B	C	A	B	C	A	B	C	
P_1	3	3	2	1	2	2	2	0	0	5	3	2	true
P_3	5	3	2	0	1	1	2	1	1	7	4	3	true
P_4	7	4	3	4	3	1	0	0	2	7	4	5	true
P_2	7	4	5	6	0	0	3	0	2	10	4	7	true
P_0	10	4	7	7	4	3	0	1	0	10	5	7	true

图 3-17 T_0 时刻的安全序列

2) P4请求资源Request4(3，3，0)，系统按银行家算法进行检查

(1) Request4(3，3，0)≤Need4(4，3，1);

(2) Request4(3，3，0)≤Available(2，3，0)，让 P4等待

3) P0请求资源Requst0(0，2，0)，系统按银行家算法进行检查

(1) Request0(0，2，0)≤Need0(7，4，3);

(2) Request0(0，2，0)≤Available(2，3，0);

(3) 系统暂时先假定可为 P0分配资源，并修改有关数据，如图 3-19

资源 / 情况 / 进　程	Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	3	0	7	3	2	2	1	0
P_1	3	0	2	0	2	0			
P_2	3	0	2	6	0	0			
P_3	2	1	1	0	1	1			
P_4	0	0	2	4	3	1			

图 3-19　为 P_0 分配资源后的有关资源数据

4) 进行安全性检查：Available(2，1，0)已不能满足任何进程的需要，故系统进入不安全状态，此时系统不再分配资源

-
- 死锁的检测与解除

一. 死锁的检测

1. 检测死锁需要：

1) 保存资源请求和资源分配信息

2) 提供算法检测是否已进入死锁状态

2. 资源分配图(Resource Allocation Graph)

1) 把结点集 N=PUR 划分为两个互斥的子集：

(1) 进程结点 P={p1，p2，…，pn}，一个圆圈表示一个进程或资源

(2) 资源结点 R={r1，r2，…，rn}，一个方块围住一些圈表示一类资源

(3) 在图 3-20 中，P={p1，p2}，R={r1，r2}， N={r1，r2}U{p1，p2}

(4)

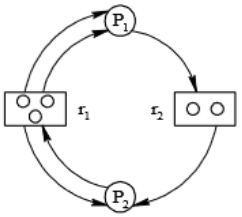


图 3-20　每类资源有多个时的情况

2) 对任意e∈E，都连接着 P 中的一个结点和 R 中的一个结点

(1) 资源请求边由进程指向资源，表示进程请求一个单位的资源

(2) 资源分配边由资源指向进程，表示把一个单位的资源分配给进程

3) 图 3-20 中，p1 进程已经分得了两个 r1 资源， 又请求一个 r2资源；p2 进程分得了一个 r1和一 个 r2资源，并又请求 r1资源

3. 死锁定理

1) 资源分配图的简化方法：

(1) 找出一个既不阻塞又非独立的进程结点 Pi。在顺利的情况下，Pi 可获得所需资源而运行完毕，再释放其所占有的全部资源，这相当于消去 pi所求的请求边和分配边，使之成为孤立的结点

(2)

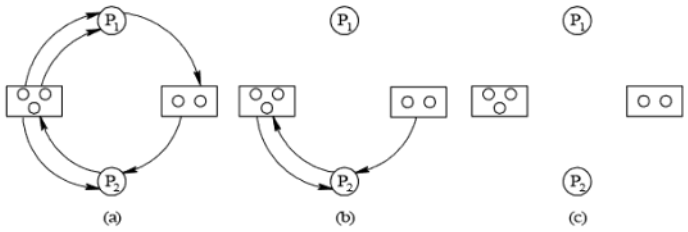


图 3-21 资源分配图的简化

(3) 循环：找下一个结点，并消去它的边

2) 若能消去图中所有的边，使所有的进程结点都成为孤立结点，则称该图是可完全简化的。可简化图无论按什么顺序都能得到同一个简化图

3) 死锁定理：死锁状态的充分条件是：资源分配图不可完全简化

4. 死锁检测中的数据结构

1) Available表示 m 类资源中每一类资源的可用数目

2) 把不占用资源的进程(向量 Allocationi:=0)记入 L 表中，即 LiUL

3) 从进程集合中找到一个 Requesti≤Work 的进程，做如下处理

(1) 简化资源分配图简化，释放资源，Work += Allocation I

(2) 将它记入 L 表中

4) 若不能把所有进程记入L表，说明将发生死锁

二. 死锁的解除

1. 解除方法：从某些进程抢占/剥夺足够资源，分配给死锁进程；终止/撤销某些进程，直至打破循环环路

1) 最简单的就是通知操作员人工处理

2) 另一类是用死锁解除算法：

2. 终止进程的方法

1) 终止所有死锁进程：简单，但代价很大，可能功亏一篑

2) 逐个终止进程，直至有足够资源

(1) 每终止一个进程，都要用死锁检测算法检查

(2) 每次要选择这些代价最小的进程终止：优先级、已执行时间、已使用资源、还需资源、交互式还是批处理式

3. 付出代价最小的死锁解除方法

1) 简单无脑但花费大的方法

(1) 先撤消进程 P1，使系统状态由 S→U1，付出的代价为 CU1

(2) 若仍处于死锁状态，需再撤消进程，直至解除死锁状态为止

(3) 再撤消进程 P2，使状态由 S→U2，其代价 为 CU2，…

(4) 可能付出的代价将是 k(k-1)(k-2)…/2C

2) 代价最小的方法：

(1) 先撤消一个死锁进程 P1，使系统状态由 S 演变成 U1

i. 将 P1记入 d(T)中，把付出代价 C1 记入 rc(T)中

ii. 对其他死锁进程重复上述过程，得到状态 U1，U2，…，Un

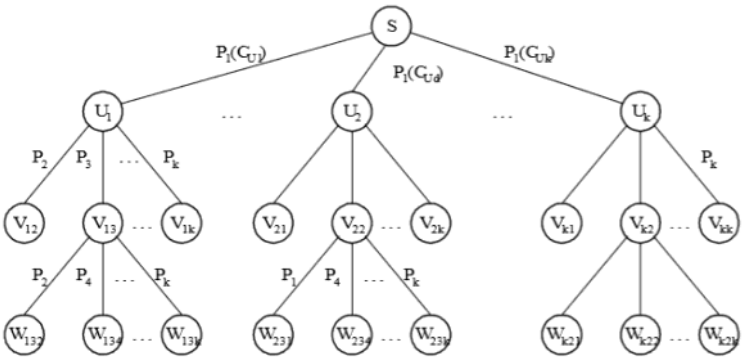
(2) 按代价大小，把进程插入到由 S 状态所演变的新状态的队列 L 中

i. 队列 L 中的首状态 U1 即为花费最小代价所演变成的状态

(3) 若仍处于死锁状态，再从 U1 状态按照上述处理方式再依次地撤消一个进程，得到 U'1，U'2，…，U'k 状态，再从 U' 状态中选取一个代价最小的 U'j，直到死锁状态解除为止

(4) 花费代价：R(S)min = min{CUi} + min{CUj} + min{CUk} + …

(5)



i.

ii.

iii.

iv.

v.

vi.

vii. --------我是底线----------

4存储器、连续分配

2018年11月3日 8:26

-
- 存储器的层次结构

1. 存储器仍是宝贵而稀缺的资源
2. 外存的文件管理与内存的管理类似，将在第7章介绍

一. 多层结构的存储器系统

1. 多层结构：最高层CPU寄存器、中间主存、最底层辅存

1)

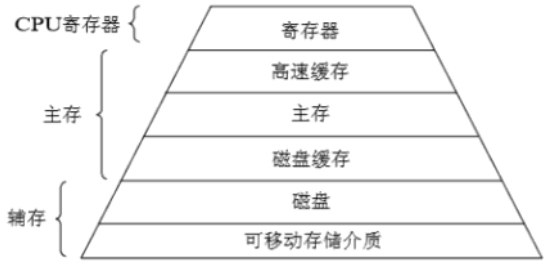


图 4-1 计算机系统存储层次示意

2) 层次越高越靠近CPU，越需要速度快，价格也越高

3) 存储器速度必须非常快才能不影响处理机的运行速度（同理I/O速度也远低于内存），因而需要缓存

4) 寄存器、主存（高速缓存、主存储器、磁盘缓存）属于操作系统管辖范畴，掉电后信息不再存在；辅存则属于设备管辖范围、存储的信息将被长期保存

2. 可执行存储器：寄存器和主存储器

1) 用load/store指令即能访问可执行存储器、而访问辅存需要通过I/O设备实现
2) 因为不涉及中断、设备驱动、物理设备、所以访问时间一般少了3个数量级

二. 寄存器和主存储器

1. 主存储器：简称内存/主存

1) 用于保存进程运行时的程序和数据，CPU与外界的交流依托其地址空间
2) 早期磁芯内存只有几十几百K，现在VLSI内存一般微机至少数十M到数G，嵌入式也有几十K到几M

2. 寄存器：与处理机同速，完全能与CPU协调工作

1) 用于存放处理机运行时的数据（操作数、地址）以加速存储器访问速度
2) 现在一般微机有数十数百个字长32或64位的寄存器；嵌入式仍不超过十几个，常为8位

三. 高速缓存和磁盘缓存

1. 高速缓存：备份主存中常用数据，减少CPU对主存的访问，大幅提高执行速度

1) 容量远大于寄存器，比内存越小两三个数量级，即几十K到几M
2) 访问速度快于主存储器，许多地方都设置了高速缓存以缓和速度矛盾
3) 程序执行的**局部性**原理：较短时间内，程序执行仅局限于某个部分；访问过的数据在短时间内会再被访问
   (1) 快要执行某段程序时，检查确定它不在高速缓存中，就临时复制进去
   (2) 下一条待执行指令也应提前准备在指令高速缓存中
4) 越快越贵，一般分几级，紧靠内存的一级最快，容量最小

2. 磁盘缓存：暂存频繁使用的部分磁盘数据和信息，减少主存访问磁盘次数

1) **磁盘缓存一般是主存中划出的一部分**，而**高速缓存是独立硬件存储器**
2) 辅存数据必须先进入主存才能给CPU用，整个主存都勉强可视作辅存缓存

3) 有些系统自动把老文件数据从辅存转储到磁带等海量存储器上，降低存储价格

-
- 程序的装入和链接

1) 用户源程序执行前，需要步骤：

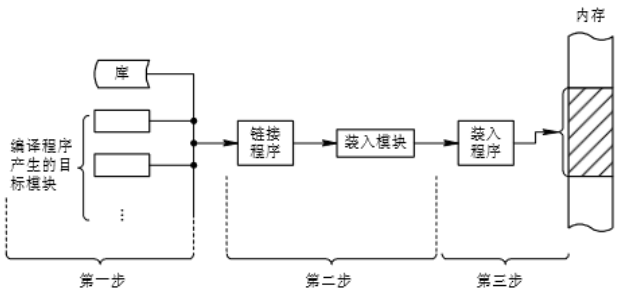
1. 由编译程序Compiler编译成若干目标模块ObjectModule
2. 由链接程序Linker将目标模块和库函数链接，形成完整装入模块LoadModule
3. 由装入程序Loader将装入模块装入内存
4. 

图 4-2 对用户程序的处理步骤

一. 程序的装入

1. 绝对装入方式(Absolute Loading Mode)
   1) 仅适用于单道系统，提前将绝对（物理）地址放入目标代码，装入程序按该地址装入
   2) 一般由编译或汇编时算出地址，也可由程序员算出并赋予。一般是编译或汇编时由符号地址转换的
2. 可重定位装入方式(Relocation Loading Mode)
   1) 多道程序不可能预知地址，只能根据内存情况装入合适位置
   2) 修改过程：将逻辑地址加上程序在内存中的首物理地址
   3) **重定位：装入时对指令和数据地址的修改过程**
   4) 静态重定位：在装入时连接装入程序一次完成重定位，以后不再改变
3. 动态运行时装入方式(Dynamic Run-time Loading)
   1) 运行过程中进程在内存中的位置可能经常要改变，使物理地址也改变
      (1) 如有对换功能的系统对一个进程多次换出换入
   2) 动态重定位：**推迟到程序执行时才重定位**，即装入内存后所有地址仍是逻辑地址
   3) 为使地址转换不影响指令执行速度，需要重定位寄存器等硬件的支持

二. 程序的链接

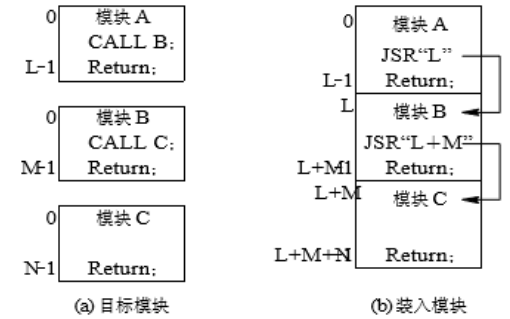
1. 静态链接方式(Static Linking)：运行前先把模块和库函数链接在一起，不再拆开
   1) 修改各模块的相对地址
   2) 变换各模块的外部调用符号为相对地址
   3) 

图 4-4 程序链接示意图
2. 装入时动态链接(Load-time Dynamic Linking)：**调用并装入模块时**修改相对地址
   1) 便于修改和更新、便于共享模块
3. 运行时动态链接(Run-time Dynamic Linking)：

1) 对某些模块的**链接推迟到程序执行时再由OS寻找并装入后**

2) 加快了程序装入过程，节省了大量内存空间

-
- 连续分配存储管理方式

一. 单一连续分配

1) 单道程序中，内存分成系统区和用户区两部分

(1) 系统区：仅供OS使用，通常放在内存低址部分

(2) 用户区：被一道程序独占

1. 存储器保护机构：防止用户程序对操作系统的破坏

1) 用户程序自己破坏操作系统后果并不严重，只是影响程序运行而已，操作系统可根据系统再启动而重新装入内存，因而当时有的电脑并没有这个机构

二. 固定分区分配

多道程序系统中，用户空间被划分为若干固定大小的区，每个分区装一道作业

1. 划分分区的方法：

1) 分区大小相等：不灵活，易浪费；控制相同对象（如炉温群控）时较方便

2) 分区大小不等：灵活，一般是较多小分区，少量大分区

2. 内存分配

1) 按分区大小建立分区使用表，包含地址、大小、分配状态

2) 分配程序按程序大小检索该表，将第一个满足的未分配区改成已分配

3) 由于大小固定，仍然可能造成浪费

4)

分区号	大小/KB	起址/KB	状态
1	12	20	已分配
2	32	32	已分配
3	64	64	已分配
4	128	128	未分配

(a) 分区说明表

(b) 存储空间分配情况

图 4-5 固定分区使用表

三. 动态分区分配/可变分区分配

1. 动态分区分配中的数据结构

1) 空闲分区表：每个空闲分区占一个表目，包括区号、大小、地址

2) 空闲分区链：一个双向链，分配后状态位由0改为1

3)

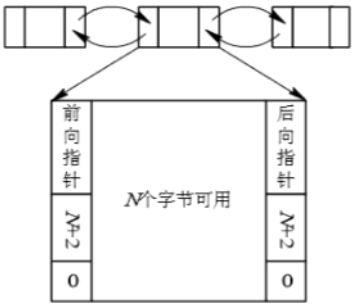


图 4-6 空闲链结构

2. 动态分区分配算法：对系统性能有很大影响。详见后几目

3. 分区分配操作：

1) 分配内存：先设置一个最小分区size

(1) 找到第一个大于内存请求的分区大小

(2) 计算分配后盈余是否大于size，是的话就只划分出一小块分区给它

(3) 然后把应分配的空间地址分给请求者，修改相关数据结构

(4)

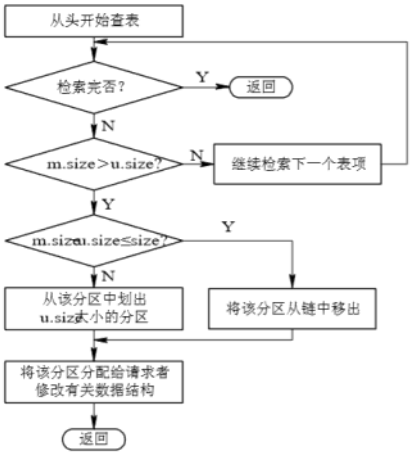


图 4-7 内存分配流程

2) 回收内存：进程运行完释放内存时，按地址找到该插入（链）表的地方

(1) 只和前一空闲分区相邻：增加前区大小

(2) 只和后一空闲分区相邻：修改后区地址为该区地址，增加后区大小

(3) 和前后空闲分区都相邻：取消后区，前区大小增加该区和后区

(4) 和前后空闲分区都不相邻：在该插的地方新建表项

四. 基于顺序搜索的动态分区分配算法

1. **首次适应**算法(first fit)：每次从链首顺序查找直到找到足够大的分区
   1) 优先利用低址分区，让后到的大作业有条件分配到大空间
   2) 低址分区被不断划分，留下很多难用的碎片，增加下次查找的开销
2. **循环首次适应**算法(next fit)：设置起始查寻指针指向上次分配区的下一个
   1) 空闲分区分布均匀，减少了查找开销
   2) 缺乏大空闲分区
3. **最佳适应**算法(best fit)：找满足要求的最小空闲分区
   1) 一般是按大小升序排列空闲分区链，也较快
   2) 容易留下难以利用的碎片
4. **最坏适应**算法(worst fit)：每次挑最大的空闲分区，割一部分下来
   1) 一般是按大小降序排列空闲分区链，缺乏大分区
   2) 出现碎片的可能性最小，对中小作业有利，查寻只需一次

五. 基于索引搜索的动态分区分配算法

1. **快速适应**算法(quick fit)/分类搜索法
   1) 将同容量的空闲分区单独设立一个链表，再在内存中设立一张管理索引表，每个表象对应一个大小的空闲分区的链表的表头指针
   2) 分配时一般不进行分割（直接把碎片分给进程）
   3) 查找效率高，一般考虑到了大空间（只是很浪费而已）
2. 伙伴系统(buddy system)
   1) 每个区的大小都是2的k次幂，同大小的空闲分区单独设立一个双向链表
      (1) 先计算恰大于等于请求的2的i次幂，并尝试找到i的双向链表
      (2) 若找不到i，则不断尝试找i++，直至找到I
      (3) 不断将I割成两个I-1伙伴块，其中一个插入I-1的表，直至割到i

(4) 回收时可能要跟伙伴块不断合并，直至没有伙伴块

2) 伙伴块地址$buddy_k(x)=x+2^k$（x%2^(k+1)==0时）

(1) 或者 $buddy_k(x)=x-2^k$（x%2^(k+1)==2^k时）

3) 时间性能差于快速适应，高于顺序搜索

4) 空间性能优于快速适应，差于顺序搜索

3. 哈希算法/散列表算法

1) 构造空闲分区大小为关键字的哈希表，每个表项记录了对应链表头指针

2) 通过哈希函数，由空闲分区大小可快速找到对应空闲分区链表

六. 动态可重定位分区分配

1. 紧凑/拼接

1) 碎片/零头：不能被利用的小分区

2) 移动内存中的作业，使他们相邻接，碎片拼接成一个大分区

3) 每次紧凑完都要对移动过的程序/数据重定位

4)

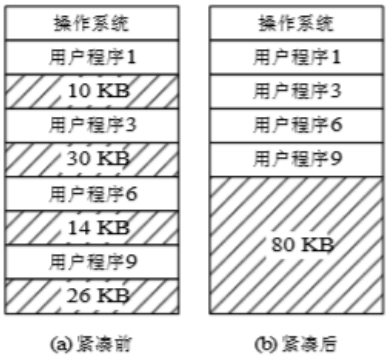


图 4-9 紧凑的示意

2. 动态重定位：地址变换过程在程序运行期间，随访问指令/数据而进行

1) 在系统中增设一个重定位寄存器，存储程序/数据在内存中的起始地址

(1) 减小地址转换对指令执行速度的影响

2) 移动指令/数据时不用对程序做修改了，只需更新内存起址

3)

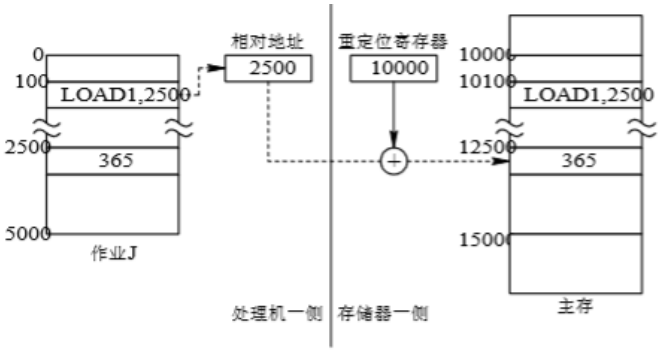


图 4-10 动态重定位示意图

3. 动态重定位分区分配算法

1) 该算法在找不到足够大的空闲分区时，若发现碎片和也小于用户要求，才返回分配失败信息；否则进行紧凑，并将拼接出的空间分配给它

2)

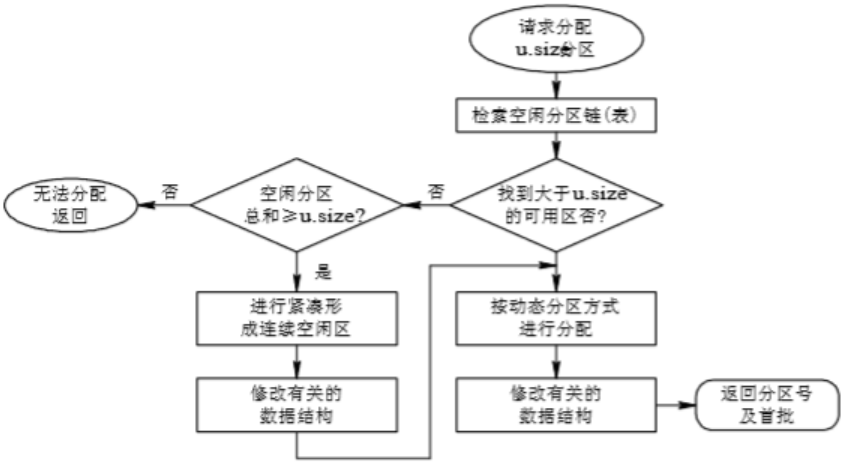


图 4-11　动态分区分配算法流程图

i.

ii.

iii.

iv.

v.

vi.

vii.

viii.

ix. ---------我是底线--------

4对换、离散分配

2018年11月12日 8:30

-
- 对换Swapping

1. 最早用于麻省理工的单用户分时系统CTSS，又称交换技术

一. 多道程序环境下的对换技术

1. 对换的引入
   1) 阻塞、内存不足时外存程序不能进入内存，浪费系统资源，降低吞吐量
   2) 增加对换设施，把内存中暂不运行的进程换到外存，把外存其他进程换入
   3) 设置对换进程，把内存中暂时不运行的进程调进磁盘对换区；把磁盘上就绪进程调入，如Windows把程序在装入内存时若发现内存不足，也会自动对换
2. 对换的类型
   1) 整体对换/进程对换/整体对换：以进程为单位，即多道程序系统中级调度
   2) 页面对换/分段对换：以页面或分段为单位，目的是支持虚拟存储系统

二. 对换空间的管理

1. 对换空间管理的主要目标：长期存储的文件区、短暂驻留的对换区
   1) 文件区占磁盘大部分，访问频率低，主要目的提高空间利用率，离散分配
   2) 对换区占磁盘小部分，访问频率高，主要目的提高出入速度，连续分配
2. 对换区空闲盘块管理中的数据结构（类似内存动态分配）
   1) 空闲分区表，每个表目含首址和大小，由盘块号和盘块数表示
3. 对换空间的分配与回收：见内存动态分配与回收

三. 进程的换出与换入

1. 进程换出
   1) 选择阻塞、睡眠态进程，再选优先级最低的、有时还考虑内存中驻留时间
   2) 换出非共享程序、非共享数据段。若传送过程未出现错误，可回收其内存空间，修改对应数据结构，直到内存中再无阻塞进程运行
   3) 一般缺页且内存紧张时才启动对换，缺页率明显减少，吞吐量下降时停止
2. 进程换入
   1) 找出就绪态的已换出进程，再找换出时间最久的进程（一般规定大于2s以上的规定时间时才换入）若申请内存成功，直接调入，失败则继续换出
   2) 一般换入成功后会紧接着找其他换出进程，直到无足够内存换入新进程

-
- 分页存储管理方式

1. 分页存储：将逻辑空间分为若干固定大小区域，称为页/页面
   1) 典型的页面大小为1KB
   2) 相应地将内存空间分为若干物理块/页框frame，块和页大小相同
2. 分段存储：将逻辑空间分为若干大小不同的段，每段定义一组相对完整的信息
3. 段页式存储：上两种相结合，具有两者的优点

一. 分页存储管理的基本方法

1. 页面和物理块

1) 页编号从0开始，如第0页、第1页；块编号从0#开始，如第0#块、第1#块

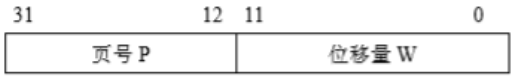
2) 进程分配时以块为单位，将若干页装入多个物理块

3) 最后一页常装不满一块，在块内形成“页内碎片”

4) 页面小，减少内存碎片空间，提高内存利用率；但页表过长，浪费内存，降低进程换进换出的效率。因而通常选1K~8K（通常是2的幂个b）

2. 地址结构

1)



2) 页号p，通常为12~31位，即一个地址空间有1M=2^20页

3) 位偏移量W，即页内地址，通常为0~11位

4) 逻辑地址空间中的地址为A，页面大小L，页内地址d，则：

(1)
$$P = INT\left[\frac{A}{L}\right]$$
$$d = [A] \ MOD \ L$$

5) 逻辑空间大小只受计算机地址位数限制

3. 页表/页面映像表：实现从页号到物理块号的地址映射

1)

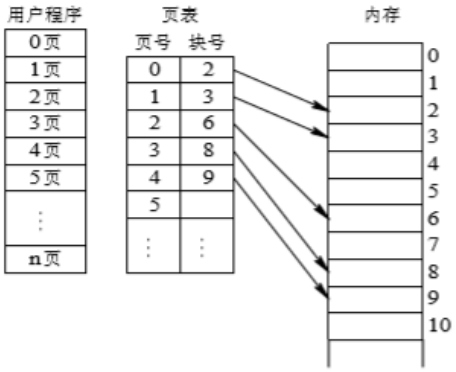


图 4-12 页表的作用

2) 存取控制字段：保护块中内容

(1) 一位字段规定内容可读写、只读

(2) 二位字段规定可读写、只读、只执行

(3) 进程试图写只读存储块时会引起操作系统一次中断

二. 地址变换机构

1) 实现从逻辑地址到物理地址的转换，由于页内地址和块内地址相同，实际上只需将逻辑地址空间页号转为内存物理块号。

2) 硬件参与的转换速度总是快于软件转换（类比交换机快于网桥）

1. 基本的地址变换机构

1) 在系统中设置页表寄存器(Page-Table Register)，存放页表始址，长度

2) 每条指令都需变换地址，但寄存器很贵，因而页表大多驻留在PCB

3) 调度进程时才更新页表寄存器，因而单处理机环境只需一个PTR

4) 变换步骤：

(1) 机构从有效地址/逻辑地址中分出页号，由硬件检索页表

(2) 若页号大于页表长度，说明越界，系统将产生地址越界中断

(3) 页表始址+页号*表项长度=该表项始址

(4) 把该表项的块号装入物理地址寄存器前几位

(5) 从有效地址/逻辑地址中分出页内地址，送入物理地址寄存器后几位

(6)

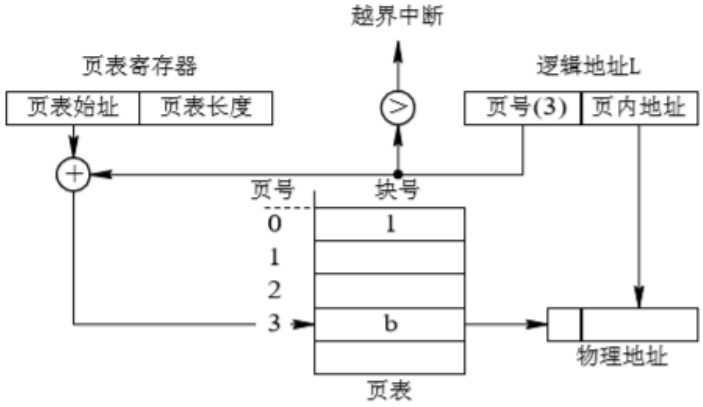


图 4-13　分页系统的地址变换机构

2. 具有快表的地址变换机构

1) 快表：为提高地址变换速度，在地址变换机构中增设一个具有并行查寻能力的特殊高速缓冲寄存器，又称为“联想寄存器”(Associative Memory)，还称Translation Lookaside Buffer

2) 新的步骤：

(1) CPU给出有效地址后，地址变换机构分出页号p，送入快表

(2) 检索出匹配页号后，直接把对应块号送入物理地址寄存器

(3) 未检索到，则重新访问内存中的页表，再加入快表

(4) 若快表已满，会挑出一个猜测不再需要的页表项，换出

(5)

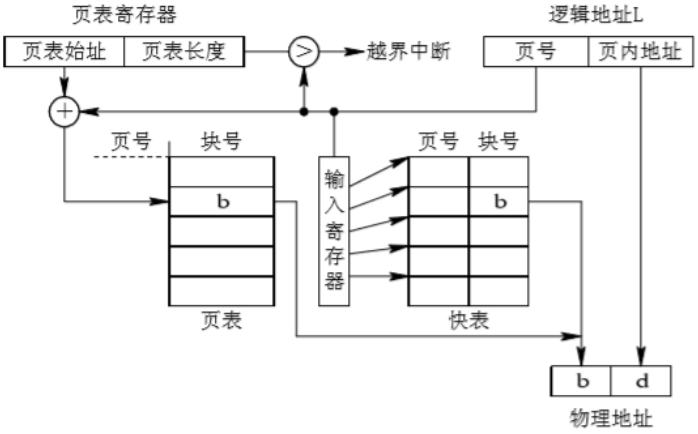


图 4-14　具有快表的地址变换机构

3) 通常快表也只放16~512个表项，但地址变换的速度损失已经可接受了

三. 访问内存的有效时间

1. 内存访问有效时间Effective Access Time：进程发出请求到操作数据所花时间
2. 无快表机构需要访问两次内存：先从页表算物理地址，再找数据
    1) 假设访问内存时间为t，则EAT=t+t=2t
3. 引入快表机构，需讨论在快表中与否，计算0-1分布数学期望
    1) 命中率a：在快表查到表项的比率
    2) 设查快表所需时间λ，则EAT=aλ+(1-a)(t+λ)=(2-a)t+aλ
    3) 例：设t=100ns，λ=20ns（纳秒，即一千分之一毫秒）

命中率a	0	50	80	90	98

有效访问时间EAT	220	170	140	130	122

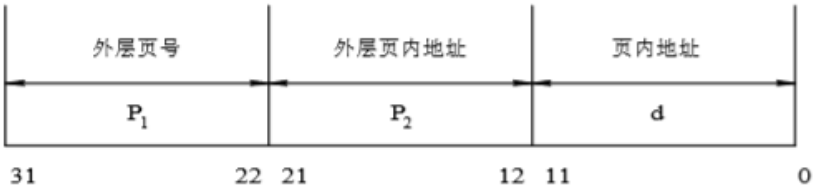
四. 两级和多级页表

1. 两级页表(Two-Level Page Table)

1) 现代计算机系统常支持逻辑地址空间2^32B~2^64B即2^2GB~2^34GB，页表将占用大量内存

2) 外层页表(Outer Page Table)：为离散分配的页表建立的页表

(1)



(2) 外层页表的每个页表项内存放的是页表分页的始址

(3) 最好再增设一个状态位S，若值为0，说明页表不在内存

3) 需增设外层页表寄存器存放外层页表的始址用外层页号做索引

4)

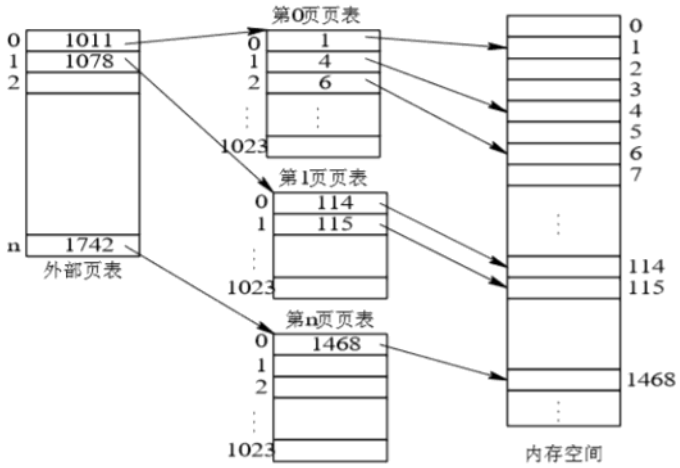


图 4-15 两级页表结构

5)

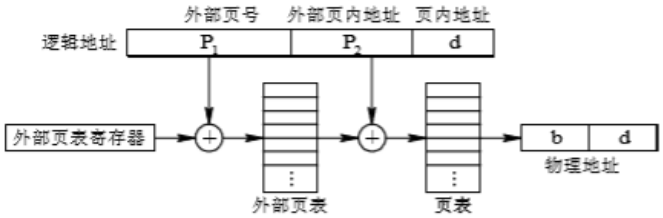


图 4-16 具有两级页表的地址变换机构

2. 多级页表

1) 64位机器的外层页号将有42位，表项将有4096G个，占空间16384G

2) 外层页表一般需要分页装入不同块，用第2级外层页表来映射外层页表

3) 64位机器支持2^64B=1844744TB规模的物理空间，也许需要很多级页表

4) 近年64位OS可直接把寻址的存储器空间减少为45位，三级页表就足够了

五. 反置页表Inverted Page Table

1. 反置页表的引入
   1) 为减少页表占用的内存空间，只为每个物理块设置一个表项，按块编号排序
      (1) 即用块编号索引，内容是页号及其隶属进程标识符
2. 地址变换
   1) 用进程标识符和页号检索反置页表，其序号i便是物理地址前几位
   2) 反置页表内容极少，未包含尚未调入内存的页面
   3) 每个进程需要一个外部页表External Page Table，调用不在内存的页面时会用到它
   4) 用进程标识符和页号检索线性表很费时，常利用Hash算法，但可能出现地址冲突

◆

◆ 分段存储管理方式

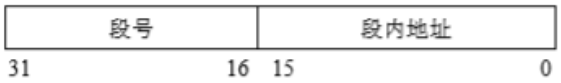
一. 分段存储管理方式的引入

1. 方便编程：按逻辑关系把作业划分成若干段
   1) 逻辑地址由段名/段号和段内偏移量/段内地址决定
      (1) LOAD 1，[A]|〈D〉将A段D单元内的值读入寄存器 1
      (2) STORE 1，[B]|〈C〉将寄存器 1 的内容存入B段C单元中
2. 信息共享：按逻辑给需要共享的一些信息建立段
3. 信息保护：同上，以逻辑单位信息为基础，标志读写属性
4. 动态增长：应对难以在运行前确定数据量的数据段
5. 动态链接：虚调用某段目标程序时，以段为单位调入目标程序并链接

二. 分段系统基本原理

1. 分段
   1) 通常用段号来代替段名，每段从0开始编址，采用一段连续的地址空间
   2) 段的长度取决于相应逻辑信息组的长度
   3) 每个段既包含地址空间，又标识了逻辑关系
   4) 逻辑地址由段号/段名和段内地址构成

   (1)



2. 段表/段映射表
   1) 记录内存始址/基址和段的长度
   2) 实现从逻辑段到物理内存的映射

   3)

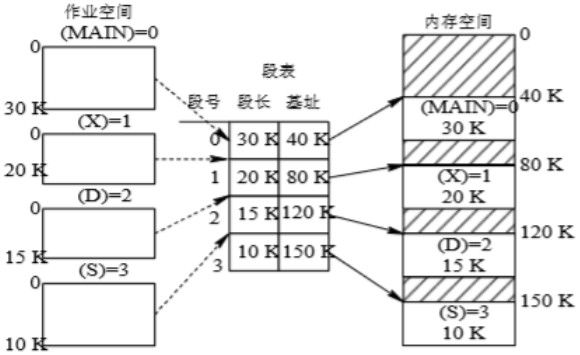


图 4-17 利用段表实现地址映射

3. 地址变换机构
   1) 段表寄存器：记录基址和表的长度TL
   2) 变换过程：
      (1) 若请求段号>TL，发出越界中断信号
      (2) 根据段表始址和请求段号，求出对应表项的位置，读出基址
      (3) 若段内地址d>段长SL，发出越界中断信号（图里漏了）
      (4) 段内地址d+=基址=需要的物理地址

(5)

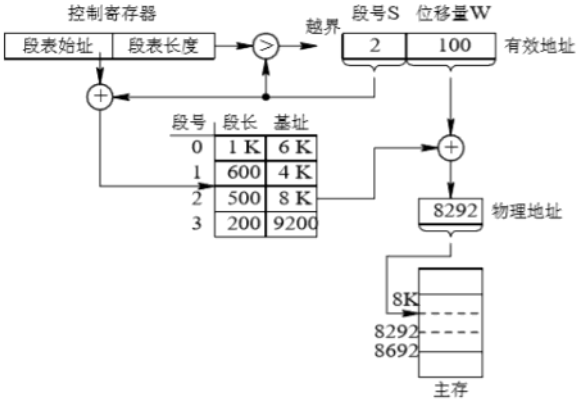


图 4-18　分段系统的地址变换过程

3) 也要访问两次内存，但段表项一般不多，所以没快表只会慢10%~15%

4. 分页vs分段：都是离散分配，都需要地址映射，但：
   1) 页是信息的物理单位，离散分布仅为提高内存利用率，是系统管理的需要
   2) 段是信息的逻辑单位，离散分布为保存一组完整信息，是用户的需要
   3) 页大小固定，由系统决定，由硬件结构划分的两部分实现
   4) 段大小不定，一般由编译程序根据信息的性质来划分
   5) 页的地址空间是一维线性的，只需一个记忆符即可表示地址
   6) 段的地址空间是**二维**的，标识地址既需段名，又需段内地址

三. 信息共享

1. 分页系统中程序和数据的共享
   1) **可重入Reentrant：能被共享**（无论在分页还是分段系统中）
      (1) 如TextEditor的，占了两个8m的ed1，ed2页和一个8m的data1页

(2)

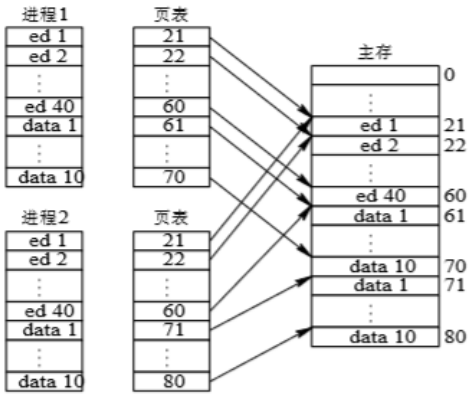


图 4-19　分页系统中共享 editor 的示意图

2. 分段系统中程序和数据的共享

1)

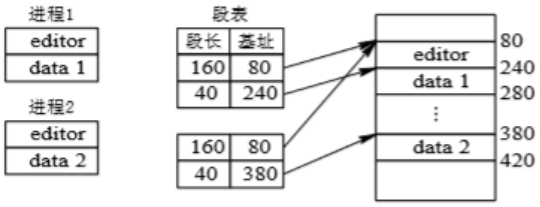


图 4-20　分段系统中共享 editor 的示意图

   2) 可重入代码Reentrant code/纯代码Pure code：能被多进程同时访问的代码
   3) 纯代码不允许在执行中做出任何改变
      (1) 若每个进程都有局部数据区，用于存储共享数据代码数据的副本，就能保证共享代码成为可重入代码

四. 段页式存储管理方式

1. 基本原理：

1) 将用户程序先分成若干个段并赋予段名，各段再分成若干个页

(1)

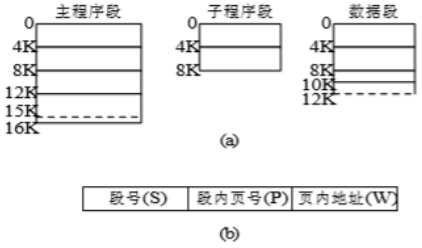


图 4-21 作业地址空间和地址结构

2) 段表项内容从内存始址和段长改为页表始址和页表长

(1)

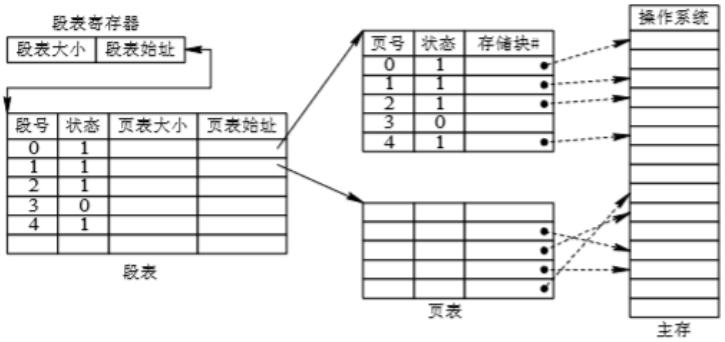


图 4-22 利用段表和页表实现地址映射

2. 地址变换过程

1) 如图

(1)

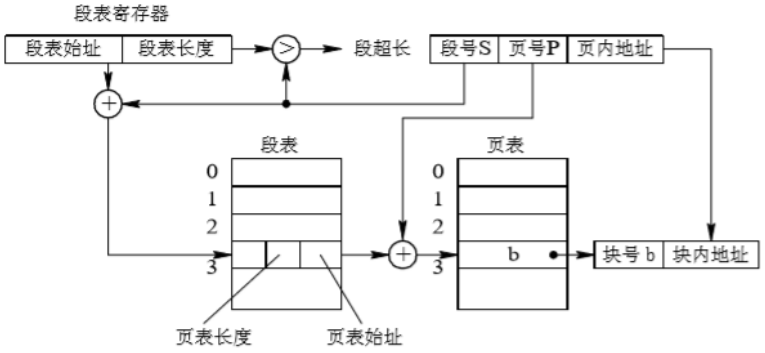


图 4-23 段页式系统中的地址变换机构

2) 访问指令/数据需要**三次访问内存：段表找页表，页表找块号，正式访问**

(1) 因而要用快表，同时利用段号和页号检索高速缓存

i.

ii.

iii.

iv.

v.

vi.

vii.

viii.

ix.

x. ---------我是底线-------

5虚拟存储器、请求分页

2018年11月12日 8:30

-
- 虚拟存储器概述

一. 常规存储器管理方式的特征和局部性原理

1. 常规存储器管理方式的特征
   1) 一次性：把作业全装入内存才能运行，限制了处理机利用率和系统吞吐量
   2) 驻留性：即使进程阻塞，不会换出任一部分
2. 局部性原理
   1) 较短时间内，程序执行仅局限于某部分
      (1) 除少部分转移和调用外，基本是顺序执行
      (2) 调用函数会使程序运行轨迹转至另一区域，但调用深度一般不超过5
      (3) 循环结构只有少数指令被多次执行
      (4) 多数据结构的处理一般局限于很小的范围内
   2) 时间局限性：短时间内，指令/数据可能被再次执行/访问，如循环
   3) 空间局限性：短时间内，访问的存储单元一般集中在一定范围内，如顺序
3. 虚拟存储器的基本工作情况
   1) 若要访问的页/段尚未调入内存，便发出缺页/段中断请求，尝试调入
   2) 若内存已满，则置换出一部分页/段，再调入进来

二. 虚拟存储器的定义和特征

1. 虚拟存储器的定义
   1) 有**请求调入功能和置换功能**，能逻辑上对内存加以扩充的存储器系统
   2) 能使用户感觉到的内存容量比实际内存容量大得多
   3) 逻辑容量取决于内、外存容量之和，运行速度近内存速度，成本近外存
   4) 内存利用率高，程序并发度高，系统吞吐量大，广泛应用于大中小微型机
2. 虚拟存储器的特征
   1) 多次性：允许将作业分多次装入内存
   2) 对换性：允许将暂不使用的代码/数据甚至整个进程调至外存
   3) 虚拟性：逻辑上扩充内存容量，能运行更大作业，提高多道程序度
3. 多次性和对换性是虚拟性的基础；离散分布是多次性和对换性的基础

三. 虚拟存储器的实现方法

1. 分页/段请求系统：在分页/段系统的基础上，**增加了请求调页/段功能和页/段置换功能所形成的的虚拟存储系统**（页和段视作两种）
   1) 允许只装入少数页/段的程序/数据即可启动
   2) 硬件支持：请求分页/段的页/段表机构、缺页/段中段机构、地址变换机构
      (1) 不少硬件被集成在处理器芯片上
   3) 软件支持：实现请求调页/段的软件和实现页/段置换的软件
      (1) 段的长度可变，因而内存分配和回收更复杂
      -
      - 请求分页存储管理方式

一. 请求分页中的硬件支持

1. 请求页表机制：

1)

页号	物理块号	状态位 P	访问字段 A	修改位 M	外存地址

2) 状态位/存在位 P：指示该页是否已调入内存，供访问时参考

3) 访问字段 A：记录本页在一段时间内被访问的次数，或记录本页最近已有多长时间未被访问，供换出时参考

4) 修改位 M：表示该页在调入内存后是否被修改过，供置换时参考

5) 外存地址：指出该页在外存上的地址，通常是物理块号，供调入时参考

2. 缺页中断机构

1) 缺页中断作为中断，也需要保护cpu环境、分析中断源、转入中断处理程序、再恢复cpu环境，但也有如下不同于其他中断的特点：

(1) 在指令执行期间产生和处理中断信号。通常，CPU 都是在一条指令执行完后，才检查是否有中断请求到达。然而，**缺页中断是在指令执行期间**，发现所要访问的指令或数据不在内存时所产生和处理的

(2) 一条指令在执行期间，可能产生多次缺页中断。如在执行一条指令 COPY A TO B 时，可能要产生 6 次缺页中断，其中指令本身跨了两个页面，A 和 B 又分别各是一个数据块，也都跨了两个页面。基于这些特征，系统中的硬件机构应能保存多次中断时的状态，并保证最后能返回到中断前产生缺页中断的指令处继续执行

i.

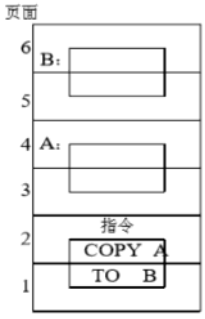


图 4-24 涉及 6 次缺页中断的指令

3. 地址变换机构

1) 分页系统地址变换机构的基础上，增加了产生和处理缺页中断，以及从内存中换出一页的功能等

(1) 先检索快表，找到待访问页后将修改位M置1

(2) 若快表中无该页表项，应去内存中找页表

(3) 在页表中根据状态位P确认该页在内存中的位置

(4) 若确认后发现尚未调入内存，应发生缺页中断，由OS调入

2)

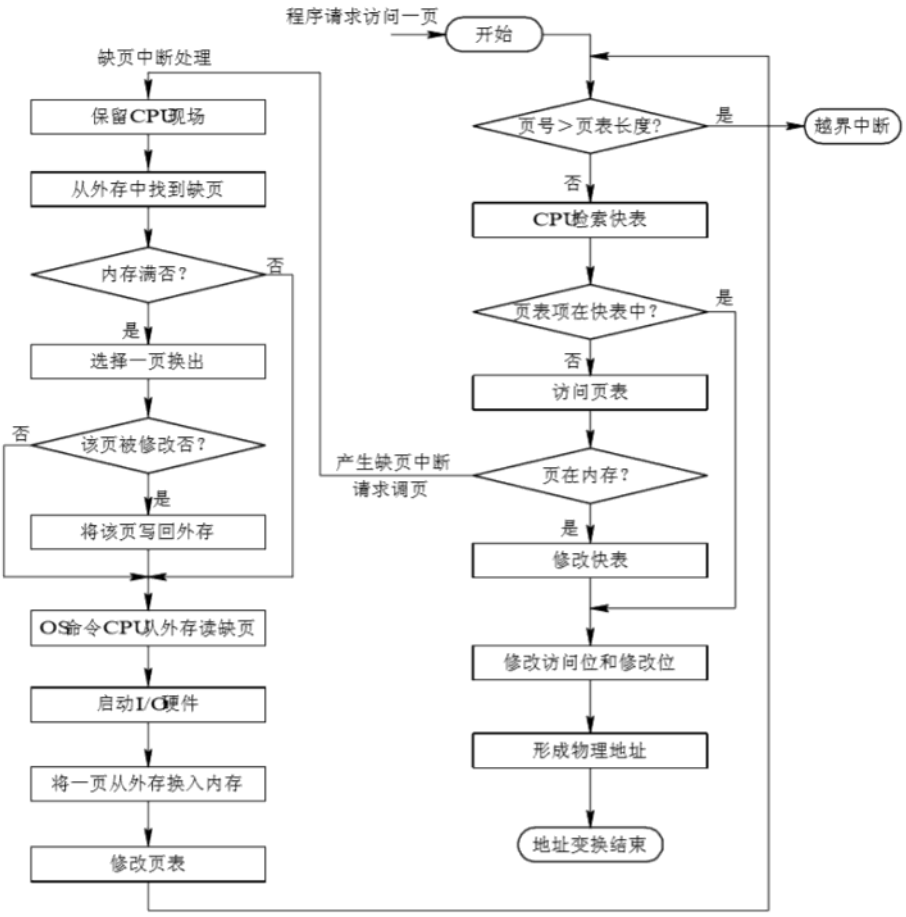


图 4-25 请求分页中的地址变换过程

(1) 左边的都是中断处理，右边的接近传统的变换机构

二. 请求分页中的内存分配

1. 最小物理块数的确定

1) 最小物理块数：能保证进程正常运行所需的最小物理块数
2) 按最小物理块数分配资源，很可能使缺页率极高，使执行速度极慢
3) 真正分配的物理块数取决于计算机硬件结构、指令格式、功能、寻址方式
   (1) 单地址指令只需2块，一个存指令。一个存数据
   (2) 允许间接寻址的机器需要至少3块
   (3) 指令长度可能有两个字节的，需要6块，见图4-25

2. 内存分配策略

(1) 固定分配：为每个进程分配固定数目物理块，运行期间不再改变
(2) 可变分配：先分配一定数目，运行期间适当增减
(3) 局部置换：发现缺页后，对页进行对换，确保分配的空间不变
(4) 全局置换：发现缺页后，取出一空闲物理块分配给该进程。若无空闲物理块，可能从其他进程选出一页做对换

1) 固定分配局部置换(Fixed Allocation，Local Replacement)
   (1) 一般根据进程类型（交互/批处理）或程序员、管理员的建议来确定
      i. 若分配太少，会频繁缺页中断，降低系统吞吐量
      ii. 若分配太多，容易使cpu等资源空闲，且进程对换时更耗时
2) 可变分配全局置换(Variable Allocation，Global Replacement)
   (1) 一般也是先根据进程类型、程序员、管理员建议来确定

(2) 全局置换时可能换出其他进程的物理块，使其缺页率增加

3) 可变分配局部置换(Variable Allocation，Local Replacement)

(1) 先根据进程类型、程序员、管理员建议，分配一些物理块

(2) 运行时再根据缺页率高/低，增加/减少一些物理块

3. 固定分配策略下的物理块分配算法

1) 平均分配：所有空闲物理块平均分配给各进程

2) 按比例分配：块数=页数/总页数*总块数（向上取整）

(1) 系统中各进程页面数的总和为：

$$S=\sum_{i=1}^{n}S_i$$

(2) 假定系统中可用的物理块总数为 m，则每个进程所能分到的物理块数为 b_i

$$b_i=\frac{S_i}{S}\times m$$

3) 考虑优先权：照顾重要的、紧迫的作业分配更多内存

(1) 一般是一部分内存按比例分配给各进程，另一部分按优先权分配

(2) 实时控制系统可能只按优先权分配物理块

三. 页面调入策略

1. 何时调入页面

1) 预调页策略：以预测为基础，将不久后可能被访问的页面调入内存

(1) 成功率一般只有一半

(2) 第一次调入进程时，可由程序员指出需调入的页

(3) 采用工作集的系统中，会将工作集中的所有页都调入内存

2) 请求调页策略：发现页不在内存时提出请求，由OS调入

(1) 易于实现，广泛采用，但花费较大开销，增加了磁盘I/O启动频率

2. 从何处调入页面

1) 将外存分为：存放文件的文件区、存放对换页面的对换区

(1) 对换区一般连续分配；文件区一般离散分配

(2) 对换区的I/O速度一般比文件区快

2) 从何处调入页面，分以下三种情况

(1) 有足够对换区空间：全部调入对换区，提高调页速度

i. 运行前需要把相关文件都拷到对换区

(2) 缺足够的对换区空间：不会修改的文件从文件区调入

i. 因为不修改的文件，在换出时可直接省去，不需修改

ii. 但可能修改的部分为了速度，仍需调到对换区

(3) UNIX方式：进程相关文件放在文件区；换出页面放在对换区

i. 未运行过的页面一般是从文件区调入

ii. 运行过（但被换出）的页面从对换区调入

iii. 因为UNIX允许页面共享，可能页面已被其他进程调入

3. 页面调入过程

1) 存在位P为0，即不在内存时，向CPU发出缺页中断

2) 中断处理程序保留cpu环境，转入缺页中断处理程序

3) 找到所在物理块后，启动I/O调入内存，修改页表

4) 若内存已满，按置换算法选出一页准备换出

5) 若换出页修改位M不是0，即被修改过，则需要将其数据写回磁盘
6) 调入所缺页后将页表中其存在位P改为0，并加入快表，
7) 之后才能根据物理地址访问内存数据，这个过程对用户透明

4. 缺页率f
   1) 设逻辑空间页数n，分配到的内存物理块数m，访问页面成功（在内存）次数S，访问页面失败次数F，总访问次数A=S+F，则f=F/A
   2) 影响因素
      (1) 页面越小，缺页率越高
      (2) 分配到的物理块越少，缺页率越高
      (3) 页面置换算法越差，缺页率越高。缺页率是衡量该算法的重要指标
      (4) 程序固有特性：局部化程度越低，缺页率越高
   3) 根据换出页被修改与否的概率，可按0-1分布数学期望计算处理时间
      i.
      ii.
      iii.
      iv.
      v.
      vi.
      vii.
      viii.
      ix. ---------我是底线-------

5页面置换、请求分段

2018年11月21日　　14:20

-
- 页面置换算法

1. 页面置换算法(Page-Replacement Algorithms)：选择换出页面的算法

一. Optimal和FIFO

1. **最佳(Optimal)**置换算法：选择以后永不使用/最长(未来)时间内不再访问的页面
   1) 由Belady于1966年提出的一种理论上的算法，通常可保证获得最低的缺页率
   2) 因为是“向后看”的，无法实现，只用来评价其它算法
   3)

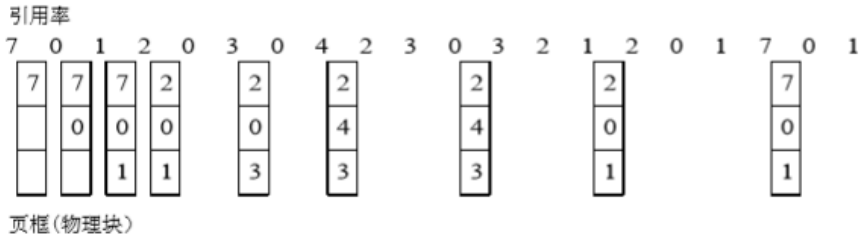


图 4-26　利用最佳页面置换算法时的置换图

2. **先进先出(FIFO)**页面置换算法：选择最先进入内存的页面/驻留时间最久的页面
   1) 是最早出现的置换算法
   2) 需要把调入内存的页面依次排成队列，设置一个替换指针指向最老的页面
   3) 可能淘汰有全局变量、常用函数、例程的页面
   4)

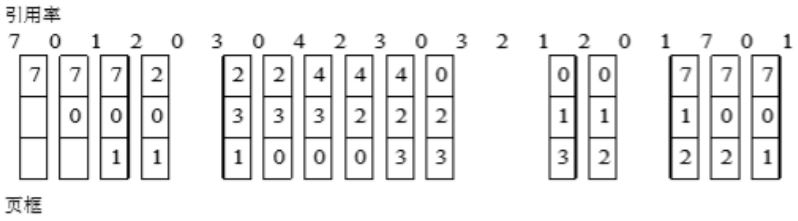


图 4-27　利用 FIFO 置换算法时的置换图

二. **最近最久未使用(Least Recently Used)**置换算法和LFU

1. LRU的描述
   1) 选择(距离上一次使用的)时间最长的页面
   2) 需要每个页面中添加一个字段记录上次使用以来的时间t
   3) 是“向前看”的较好算法，不过过去和未来的走向并无必然联系
   4)

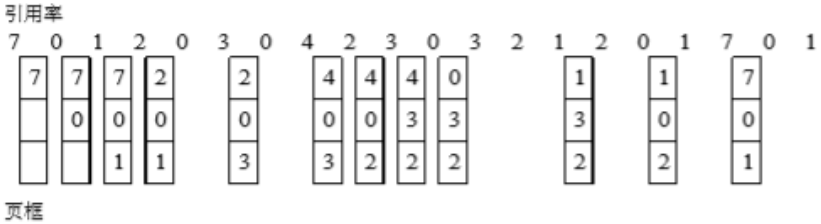


图 4-28　LRU 页面置换算法

2. LRU的硬件支持
   1) 寄存器
      (1) $R = R_{n-1}R_{n-2}R_{n-3} \cdots R_2R_1R_0$
      (2) 访问某物理块时将其R_{n-1}置1

(3) 每隔一段时间（如100ms）将所有寄存器右移一位

(4) 最小二进制数值的寄存器对应的页面就是淘汰页面，如第3行

(5)

实页 \ R	R_7	R_6	R_5	R_4	R_3	R_2	R_1	R_0
1	0	1	0	1	0	0	1	0
2	1	0	1	0	1	1	0	0
3	0	0	0	0	0	1	0	0
4	0	1	1	0	1	0	1	1
5	1	1	0	1	0	1	1	0
6	0	0	1	0	1	0	1	1
7	0	0	0	0	0	1	1	1
8	0	1	1	0	1	1	0	1

图 4-29 某进程具有 8 个页面时的 LRU 访问情况

2) 栈

(1) 每当访问某页面时，将其页面号移出，压入栈顶

(2) 栈底的就是淘汰页面

(3)

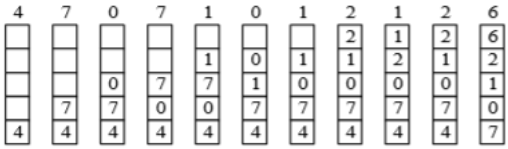


图 4-30 用栈保存当前使用页面时栈的变化情况

3. **最少使用**(Least Frequently Used)置换算法：选择最近时期内使用最少的页

1) 1ms可能访问页面成千上万次，因而难以实现记录访问次数

2) 只能用LRU寄存器反映某段时间为单位的使用频率

3) 并不能真实反映使用频率，因为时间间隔内访问1次和1000次是等效的

三. **Clock置换/最近未用**(Not Recently Used)

1. 简单的Clock置换

1) 每页设置一访问位，被访问后置1

2) 淘汰时依次找访问位0，并把1置0

3) 内存中的页面被保存成循环链表，因而全为1时会回到第一个做淘汰

4)

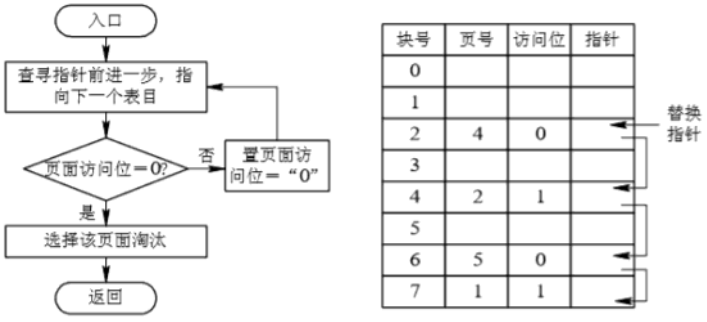


图 4-31 简单 Clock 置换算法的流程和示例

2. 改进型Clock置换

1) 除了访问位A以外，还要考虑修改位M

(1) A=0，M=0：该页最近既未被访问，又未被修改，是最佳淘汰页

(2) A=0，M=1：该页最近未被访问，但已被修改，并不是很好的淘汰页

(3) A=1，M=0：该页最近已被访问，但未被修改，该页有可能再被访问

(4) A=1，M=1：该页最近已被访问且被修改，该页可能再被访问

2) 执行过程
   (1) 找A=0且M=0的页面淘汰，（此次查找不改访问位A）
   (2) 找A=0而M=1的页面淘汰，并将此次查找过程经过的A置0
   (3) 回到第一步
3) 尽量减少磁盘I/O次数，但扫描次数可能多了
4) 是UNIX的算法

四. **页面缓冲**算法(Page Buffering Algorithm)

1. 影响页面换进换出效率的若干因素
   1) 页面置换算法，影响了缺页率
   2) 写回磁盘的频率。如把已修改换出页面挂在一个链表上，暂时不写进磁盘，等表长到一定数值时一起写回，就能减少磁盘I/O次数和换出开销
   3) 读入内存的频率。如果未写回磁盘时又想访问已修改页面，可直接从修改换出页面链表上取下，就能减少读入内存的频率，减少换进开销
2. 页面缓冲算法PBA
   1) 为降低页面换进换出频率，VAX/VMS系统中设置了：
      (1) 空闲页面链表，链接了空闲物理块
         i. 读入的页面可以装入该表第一个物理块
         ii. 未修改的换出页面都挂在其后，下次想读，可以直接取下
      (2) 修改页面链表，链接了已修改待换出页面
         i. 想换出已修改的页面时，将该物理块挂在其后
   2) 特点：
      (1) 显著降低页面换进换出频率，减少磁盘I/O次数，减少换进换出开销
      (2) 因换进换出开销大幅减少，可用FIFO等不需硬件支持的置换策略

五. 访问内存的有效时间EAT

1. 需要讨论页表项是否在快表中，不在快表还要讨论页表项在内存中与否，分别用0-1分布数学期望

-
- “抖动”与工作集

1. 抖动(Thrashing)：刚被换出的页面很快又被访问。运行时间大部分花在对换

一. 多道程序度与抖动

1. 多道程序度与处理机的利用率
   1) 利用率一开始随多道程序数增加而急增，进程数到达Nmax后利用率达到最高，之后缓慢下降，进程数多都某一点后利用率趋于0，就是因为抖动
2. 产生抖动的原因：系统中同时运行的进程太多，分配给每个进程的物理块数太少，不能满足进程正常运行的基本要求，使进程频繁缺页，使系统中排队等待页面调进调出的进程数目增加
   1) 因大部分时间用于换进换出，磁盘的有效访问时间也急剧增加

二. 工作集

1. 工作集的基本概念
   1) 1986Denning提出：程序运行的某段时间内，仅访问一部分页面
   2) 称这部分页面为活跃页面
2. 工作集：某段时间间隔Δ内，进程实际需要访问的页面的集合
   1) 把时间t内的工作集记为w(t,Δ)，Δ称为工作集的窗口尺寸windows size
   2) 工作集又可定义为进程在时间间隔(t-Δ,t)中引用的页面的集合
   3) w(t,Δ)是二元函数，是Δ 的非降函数nondecreasing function
      (1) 即$w(t,\Delta) \subseteq w(t,\Delta+1)$

三. 抖动的预防方法

1. 局部置换

1) 不允许从其他进程获得新物理块，因而抖动被限制在小范围，不影响其他进程
2) 然而抖动进程会长期处于磁盘I/O等待队列，延长其他进程缺页中断处理时间，间接延长其他进程访问磁盘时间

2. 把工作集算法融入到处理机调度中
   1) 调度程序从外存调入作业前，先检查各进程在内存中驻留页面是否够多
   2) 为缺页率高的作业增加新物理块，此时不再调入新作业
3. 利用L=S准则调节缺页率
   1) L：发生两次缺页之间的平均时间
   2) S：平均缺页服务时间，即置换一个页面所需的时间
   3) L远>S说明很少缺页，磁盘能力尚未被充分利用
   4) L<S说明频繁发生缺页，且缺页速度已超过磁盘处理能力
   5) L接近S时，磁盘和处理机才能达到最大利用率
4. 选择暂停进程：即减少多道程序的数目
   1) 选择策略：优先级低、并不重要、占块较多、剩余执行时间最多的块

- ◆
- ◆ 请求分段存储管理方式

一. 请求分段中的硬件支持

1. 请求段表机制

1)

段名	段长	段的基址	存取方式	访问字段A	修改位M	存在位P	增补位	外存始址

2) 存取方式。若为两位，则可有只执行、只读、可读写
3) 访问字段A：访问频繁度。供置换算法参考
4) 修改位M：在内存中是否被修改过、供置换页面时参考
5) 存在位P：是否已调入内存。供程序访问时参考
6) 增补位：是否做过动态增长。是请求分段式管理中特有的字段
7) 外存始址：本段在外存中的始址，即起始盘块号

2. 缺段中断机构

1) 类似缺页中断，不过段不定长，稍复杂

2)

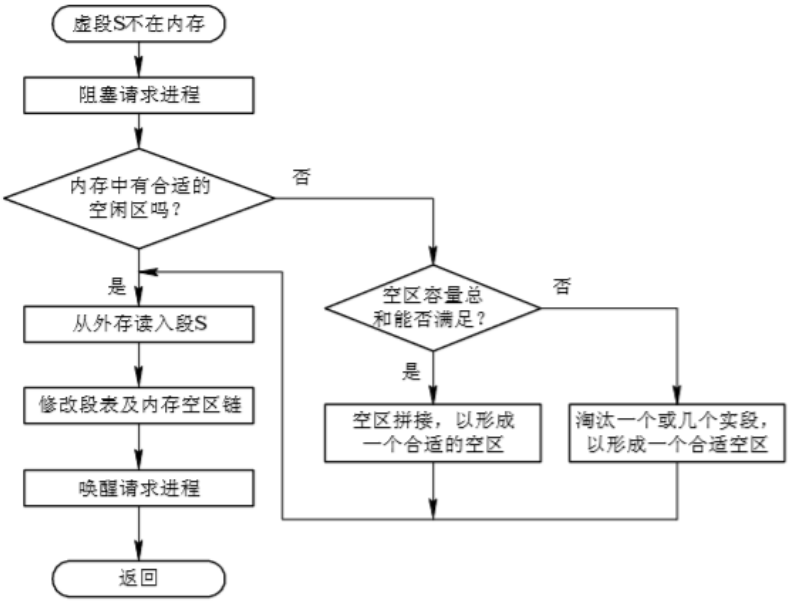


图 4-32 请求分段系统中的中断处理过程

3. 地址变换机构

1) 若要访问段不在内存，必须先调入内存，并修改段表，才能利用段表做地址变换。在传统的地址变换中增加了缺段处理，分段保护处理等

(1)

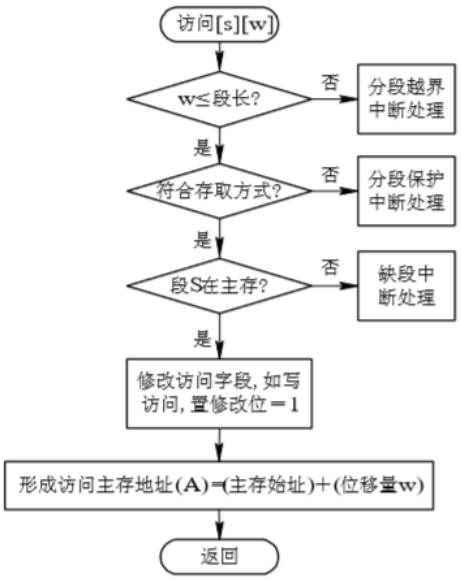


图 4-33　请求分段系统的地址变换过程

二. 分段的共享与保护

1. 共享段表

1)

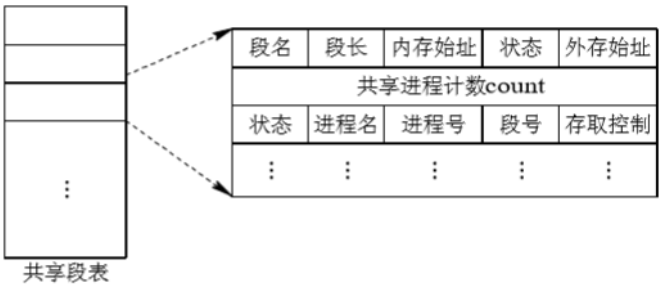


图 4-34　共享段表项

2) 共享计数count：有多少进程在共享该分段

3) 存取控制字段：只执行/只读/可读写

4) 段号：不同进程访问该共享段用到的不同段号

2. 共享段的分配与回收

1) 共享段的分配：

(1) 第一次请求段时在共享段表中增加它的表项，count置1

(2) 之后其他请求段在表项下面填写新一行，count++

2) 共享段的回收

(1) 删除对应表项，count--

(2) 若count减为0，需由系统回收其物理内存，取消其表项

3. 分段保护

1) 越界检查

(1) 利用地址变换机构完成

(2) 若段号大于段表长度，将发出地址越界中断信号

(3) 若段内地址大于段长，也发出地址越界中断信号

2) 存取控制检查

i. 只执行：只能调用该段，不能读内容，不能修改

ii. 只读：只能访问该段的程序/数据

iii. 读/写：运行读，运行写

(1) 既要考虑信息安全，又要满足运行需要

(2) 是基于硬件实现的

3) 环保护机构

(1) 低编号的环有高优先权。如OS核心处于0号环，重要的实用程序和操作系统服务占居中间环，一般应用程序在外环

(2) 程序可以访问驻留在同环或低特权外环的数据

(3) 程序可以调用驻留在同环或搞特权内环的服务

(4)

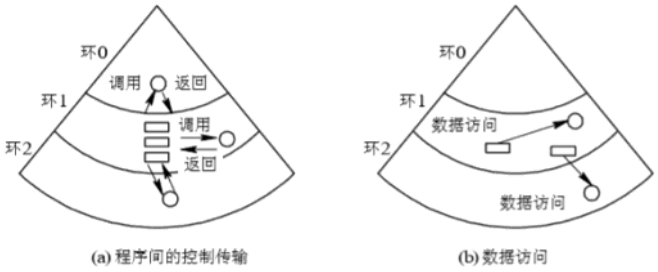


图 4-35 环保护机构

i.

ii.

iii.

iv.

v.

vi.

vii.

viii.

ix.

x.

xi. ---------我是底线-------

6IO系统、设备

2018年11月21日 15:04

-
- I/O系统的功能、模型和接口

一. I/O系统的基本功能：方便IO、提高设备利用率、准确无误（之后省略I/O的斜杠）

1. 隐藏物理设备的细节
   1) 不同IO设备在接收/产生数据速度、传输方向、粒度（数据细化程度）、数据表示形式、可靠性等方面都有很多差异
   2) 各设备应配置相应设备控制器（有若干命令/参数寄存器的硬件）
   3) 用户通过命令和参数控制外部设备执行相应操作
2. 与设备的无关性
   1) 隐藏物理设备细节的基础上，用抽象的逻辑设备名来使用设备。如只提供读写命令和逻辑设备名，由系统自己安排打印机
   2) 有效提高了系统的可移植性和易适应性，即不需重新编译系统即可添加新设备驱动程序。如windows即插即用IO设备
3. 提高处理机和IO设备的利用率
   1) 尽可能让处理机和IO设备并行操作
   2) 处理机应尽快相应IO请求，尽快运行IO设备
   3) 尽量减少IO设备运行时处理机的干预时间
4. 对IO设备进行控制
   1) IO软件应屏蔽差异，给高层软件和用户提供统一的方便操作接口
   2) 轮询的可编程IO方式：不断测试忙闲标志
   3) 中断的可编程IO方式：打印机，键盘等传输单位为字节/字的低速设备
   4) 直接存储器访问方式：磁盘、光盘等传输单位为数据块的高速设备
   5) IO通道方式：使IO操作的组织、数据的传输能无CPU干涉地独立进行
5. 确保对设备的正确共享
   1) 共享设备：运行多进程同时访问的设备，可交叉进行读写，不影响正确性
   2) 独占设备：应互斥访问的设备，分配给某进程后，直至用完才释放，因而需要考虑分配的安全性。如打印机、磁盘机
6. 错误处理
   1) 临时性错误可通过重试来纠正
   2) 重试多次仍无法解决将被视作持久性错误
   3) 持久性错误一般是与设备紧密相关的，尽可能在接近硬件的层面上解决
   4) 低层软件实在解决不了的错误才向上层报告并请求解决

二. IO系统的层次结构和模型

每一层都利用下层提供的服务，完成输入输出的某些子概念，屏蔽功能实现细节，向高层提供服务

1. IO软件的层次结构

1)

2) 用户层IO软件：实现与用户交互的接口，用户可直接调用该层的库函数
3) 设备独立性软件：实现用户程序与设备驱动器的统一接口、设备命名、设备保护、设备分配与示范等，同时为设备管理和数据传送提供存储空间
4) 设备驱动程序：与硬件直接相关，实现操作指令，驱动设备工作
5) 中断处理程序：保持CPU环境、处理中断、恢复现场，回到进程

2. IO系统各模块间的层次视图

(1)

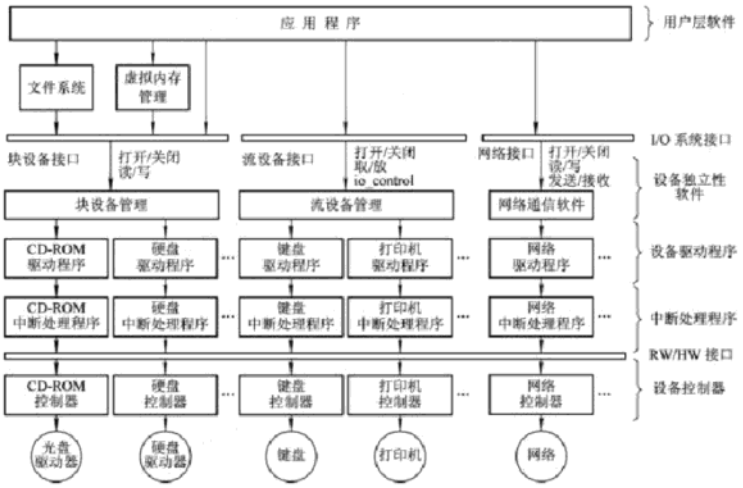


图6-2 I/O系统中各种模块之间的层次视图

1) IO系统的上下接口
   (1) IO系统接口：IO系统与上层系统（文件系统、虚拟存储器、进程）间的接口，提供抽象IO命令
   (2) 软件/硬件(RW/HW)接口：上层是中断处理、设备驱动，下层是设备控制器（CD-ROM、硬盘、键盘、打印机、网络等的控制器）
2) IO系统的分层
   (1) 中断处理程序：最底层，与硬件直接交互，当IO设备发出中断请求后，做出初步处理便转向中断处理程序。保存环境、转入处理、恢复环境、返回断点
   (2) 设备驱动程序：次底层，是进程和设备控制器之间的通信程序。把上层抽象IO请求转换为具体命令和参数，并装入设备控制器的寄存器中，或相反。每次增加新设备到系统，都要安装新驱动
   (3) 设备独立性软件/设备无关性软件：独立于具体使用物理设备的IO软件，提高了系统的可适应性和可扩展性。包括设备命名、设备分配、数据缓冲、数据高速缓冲等软件

三. IO系统接口

1. 块设备接口：块设备管理程序与高层间的接口，控制其输入输出
    1) 块设备：以数据块为存取传输单位的设备，如磁盘
        (1) 传输效率较高，数m到数十m每秒，可寻址，常用DMA方式IO
    2) 隐蔽磁盘二维结构，将扇区从0到n-1编号，用磁道号和扇区号线性表示扇区地址
    3) 将抽象命令映射为低层操作
    4) 如缺页中断后，IO系统通过块设备接口从磁盘将所缺页调入内存
2. 流设备接口：流设备管理程序与高层间的接口，控制其输入输出
    1) 流设备/字符设备：以字符为单位，如键盘、打印机
        (1) 传输效率低，数到数千字节每秒，不可寻址，常用中断驱动方式IO
    2) get/put操作：在字符缓冲区，顺序存取字符
    3) in-control指令，包含了许多参数，对应具体设备相关的特定功能
    4) 基本都是独占设备，需要互斥共享，使用前后需要打开/关闭
3. 网络通信接口：通过网络与网络上其他计算机通信或上网浏览

-
- IO设备和设备控制器

1. 执行IO操作的机械部分就是一般IO设备，执行控制IO电子部件则成为设备控制器或适配器adapter
2. 微、小型机中的控制器一般是插入扩展槽中的印刷电路卡，因此又称控制卡、接口卡、网卡
3. 中、大型机一般配置IO通道或IO处理机

一. IO设备

1. IO设备类型
    1) 按使用特性分类
        (1) 存储设备/外存/后备存储器/辅助存储器，是存储信息的主要设备。存取速度较内存慢，但容量大得多，价格便宜
        (2) 输入设备、输出设备、交互式设备
            i. 输入设备接收外部信息，如键盘、鼠标、扫描仪、视频摄像、 各类传感器等
            ii. 输出设备将信息送向外部，如打印机、 绘图仪、显示器、音响等
            iii. 交互式设备集成上述两类
    2) 按传输速率分类
        (1) 低速设备：传输速率仅每秒几字节至数百字节。键盘、鼠标器、语音的输入和输出等
        (2) 中速设备：传输速率在每秒数千字节至数十万字节。行式打印机、 激光打印机等
        (3) 高速设备：传输速率在数百千字节至千兆字节。磁带机、磁盘机、光盘机等
    3) 按信息交换的单位分类：块设备、字符设备
    4) 按共享属性分类：独占、共享、虚拟设备
2. 设备与控制器之间的接口：设备通常不直接与 CPU 进行通信，只跟控制器通信
    1) 数据信号线：在设备及其控制器之间传送数据信号
    2) 控制信号线：控制器向IO设备发送控制信号时的通路
    3) 状态信号线：传送设备当前状态信号的线

4)

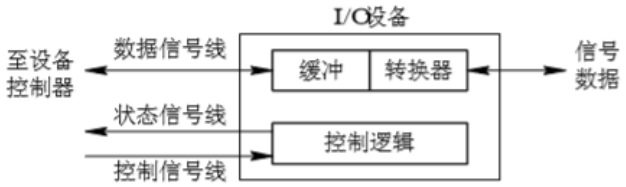


图 5-1　设备与控制器间的接口

二. 设备控制器：设备与CPU之间的接口

1. 设备控制器基本功能
   1) 接收和识别命令：用控制寄存器和命令译码器，存放接收CPU的命令和参数，并进行译码
   2) 数据交换：用数据寄存器和信号线实现 CPU 与控制器之间、控制器与设备之间的数据交换
   3) 标识和报告设备的状态：用状态寄存器供 CPU 了解设备状态
   4) 地址识别：用地址译码器识别每个设备对应的唯一地址
   5) 数据缓冲：用缓冲器减缓设备与CPU或内存的速率不匹配问题
   6) 差错控制：将差错检测码置位，并报告CPU将本次传送来的数据作废，并重新进行一次传送。保证数据输入的正确性
2. 设备控制器的组成
   1) 设备控制器与处理机的接口：数据线、地址线和控制线
      (1) 数据线通常与数据寄存器或控制/状态寄存器相连接
   2) 设备控制器与设备的接口
      (1) 每个接口中都存在数据、控制和状态三种类型的信号
   3) I/O 逻辑：实现对设备的控制
      (1) 处理机通过一组控制线利用该逻辑向控制器发送 I/O 命令
      (2) I/O 逻辑对收到的命令进行译码，再对所选设备进行控制
      (3) 每当 CPU 要启动一个设备时，一方面将启动命令发送给控制器；另一方面又同时通过地址线把地址发送给控制器，由 I/O 逻辑对收到的地址进行译码

(4)

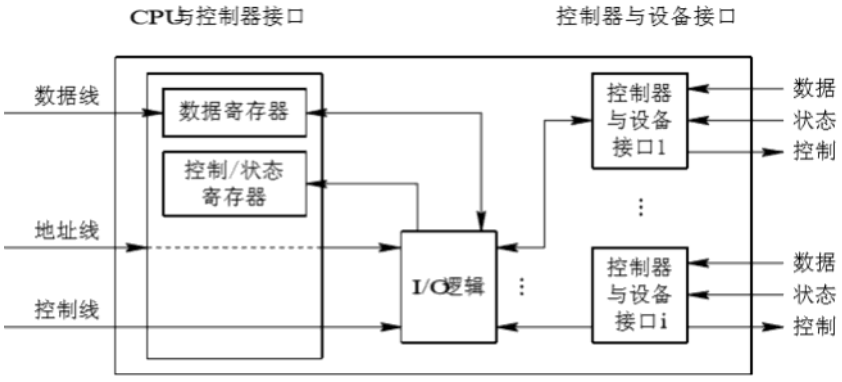


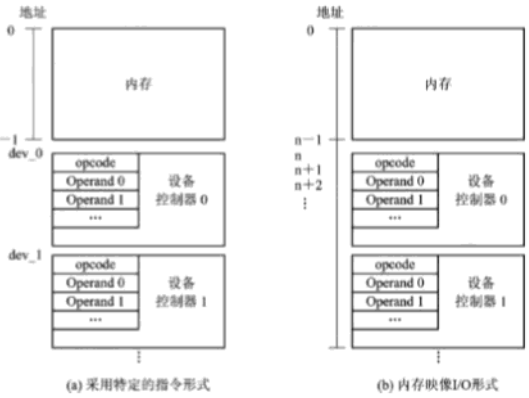
图 5-2　设备控制器的组成

三. 内存映像IO：将抽象IO命令转换出的命令/参数装入控制器的寄存器

1. 利用特定的IO指令
   1) 如Store cpu-reg k将cpu的reg寄存器的数据存入内存中的k单元
   2) 访问设备可能需要不同的指令，主要指单元记法不同
2. 内存映像IO
   1) 编址上不再区分内存单元地址和设备控制器中的寄存器地址，都采用k

2) 当k大于等于n时被认为是寄存器址（因为内存地址从0计，最大值为n）

3)



四. IO通道

1. IO通道设备的引入
   1) 目的是使IO操作不仅在数据传送，而且在IO操作组织、管理、结束处理都尽量独立于CPU
   2) **IO通道可视作特殊的处理机**，有执行IO指令的能力，特殊在：
      (1) 指令类型单一，只执行与IO操作有关的指令
      (2) 没有自己的内存，只借用主机内存，即与CPU共享内存
2. 通道类型
   1) 字节多路通道(Byte Multiplexor Channel)
      (1) 按字节交叉方式工作
      (2) 通常都有许多非分配型子通道，数量从几十到数百个，每个子通道连接一台 I/O 设备，并控制其 I/O 操作
      (3) 子通道按时间片轮转方式共享主通道
      (4) 不适于连接高速设备，会容易丢失信息

(5)

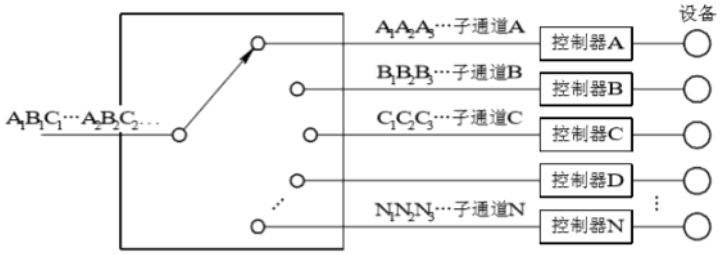


图 5-3 字节多路通道的工作原理

   2) 数组选择通道(Block Selector Channel)
      (1) 按数组方式传送，只有一个分配性子通道，一段时间内只能执行一道通道程序，控制一台设备，直至被释放才能供下一设备使用
   3) 数组多路通道(Block Multiplexor Channel)
      (1) 两者结合，多个非分配型字通道，按数组方式传送
      (2) 高传输速率和高通道利用率，广泛用于多台高中速外围设备系统
3. 瓶颈问题：通道价格昂贵，数量势必较少，进而造成系统吞吐量下降
   1) 单通路方式：如图，设备1、2、3任一个使用时，4就不能使用

(1)

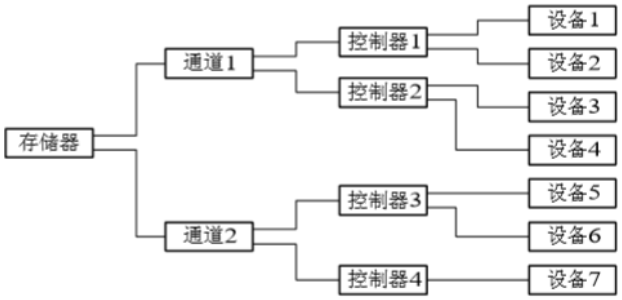


图 5-4　单通路 I/O 系统

2) 多通路方式：增加设备到主机间的通路而不增加通道

(1) 即把一个设备连上多个控制器

(2) 不仅解决了瓶颈问题，还提高了系统可靠性，个别通道或控制器故障不会导致设备和存储器间无通路

(3)

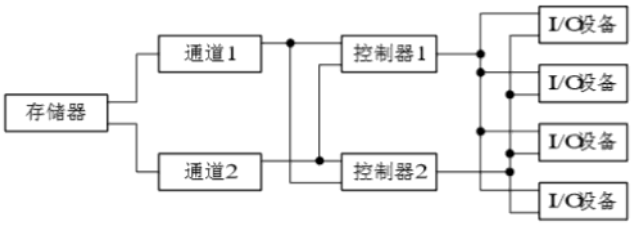


图 5-5　多通路 I/O 系统

五. 总线系统：实现 CPU、存储器以及各种 I/O 设备之 间的联系的系统

1. ISA 和 EISA 总线

1) ISA(Industry Standard Architecture)总线：1984 年推出的微机的总线结构。其带宽为 8 位，最高传输速率为 2 Mb/s。之后不久又推出了 16 位的(EISA)总线，其最高传输速率为 8 Mb/s，后又升至 16 Mb/s，能连接 12 台设备

2) EISA(Extended ISA)总线：20 世纪 80 年代末期，为满足带宽和传输速率的要求，开发出扩展 ISA(EISA)总线，其带宽为 32 位，总线的传输速率高达 32 Mb/s，同样可以连接 12 台外部设备

3)

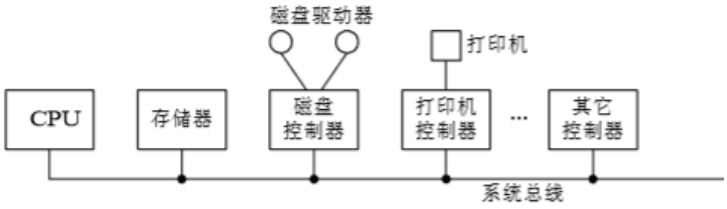


图 5-6　总线型 I/O 系统结构

2. 局部总线(Local Bus)

(1) 多媒体技术的兴起，特别是全运动视频处理、高保真音响、高速 LAN，以及高质量图形处理等技术，都要求总线具有更高的传输速率，于是，局部总线便应运而生

(2) 局部总线：将多媒体卡、高速 LAN 网卡、高性能图形板等，从 ISA 总线上卸下来，再通过局部总线控制器直接接到 CPU 总线上，使之与高速 CPU 总线相匹配，而打印机、FAX/Modem、CDROM 等仍挂在 ISA 总线上。在局部总线中较有影响的是 VESA 总线和 PCI 总线

1) VESA(Video Electronic Standard Association)总线

(1) 其设计思想是以低价位迅速占领市场。带宽为 32 位，最高传输速 率为 132 Mb/s。它在 20 世纪 90 年代初被推出时，广泛应用于486微机中。仍存在较严重的缺点，如，能连接的设备数仅为 2 ~ 4 台，在控制器中无缓冲，难于适应处理器速度的不断提高，也不能支持后来出现的 Pentium 微机

2) PCI(Peripheral Component Interface)总线

(1) 随着 Pentium 系列芯片的推出，Intel 公司颁布了 PCI 总线的 V1.0 和 V2.1 规范，后者支持 64 位系统。PCI 在 CPU 和外设间插入一复杂的管理层，协调数据传输和提供一致的接口。在管理层中配有数据缓冲，通过该缓冲可将线路的驱动能力放大，使 PCI 最多能支持 10 种外设，并使高时钟频率的 CPU 能很好地运行，最大传输速率可达 132 Mb/s。PCI 既可连接 ISA、EISA 等传统型总线，又可支持 Pentium 的 64 位 系统，是基于奔腾等新一代微处理器而发展的总线

i.
ii.
iii.
iv.
v. ----------我是底线-------

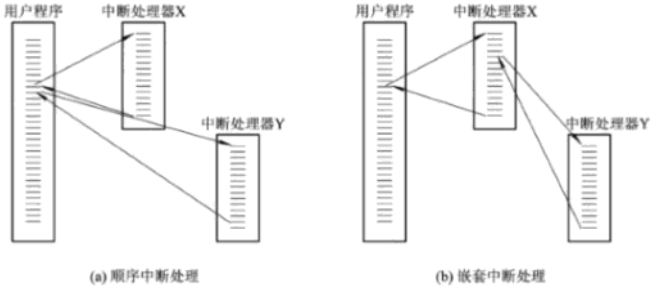
6中断、驱动、软件

2018年12月17日 10:45

-
- 中断机构和中断处理程序

一. 中断简介

1. 中断和陷入
   1) 中断/外中断：CPU对外部IO设备发来的中断信号的响应
   2) 陷入trap/内中断：CPU内部时间引起的中断，如运算溢出、地址越界、电源故障
2. 中断向量表和中断优先级
   1) 中断向量表：为每种设备匹配以相应的中断处理程序，把其入口放在一个表项，为每个中断请求规定一个中断号，对应一个表项。由中断控制器根据中断号查找中断向量表，再转入中断处理程序
   2) 中断优先级：根据服务紧急度给不同中断信号源排级，如键盘<打印机<磁盘
3. 对多中断源的处理方式
   1) 屏蔽/禁止中断：处理一个中断时让所有新到中断请求都等待
   2) 嵌套中断：优先响应高优先级的中断请求，高级的可以抢占低的处理机
   3)



二. 中断处理程序

1. 请求IO操作的进程会被挂起，IO完成后，设备控制器会向cpu发送中断请求
2. 中断处理步骤
   1) 测定是否有未响应的中断信号。每当设备完成字符操作，设备控制器便发出中断请求，将数据在设备和内存缓冲之间交换。因此cpu每执行完一条指令都要测定是否有中断
   2) 保护被中断进程的cpu环境。把处理机状态字psw和程序计数器pc的下条指令的地址存到中断保留栈，把cpu寄存器内容压入中断栈

(1)

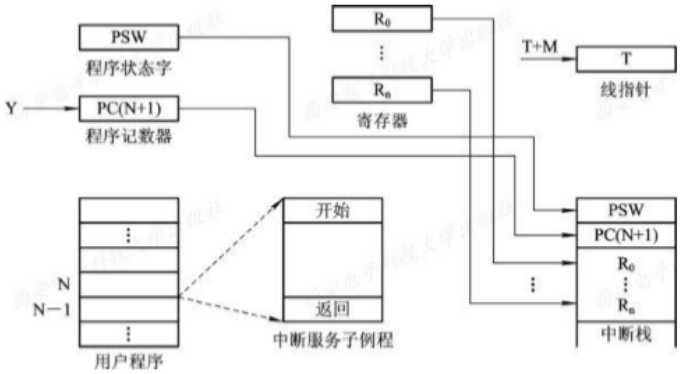


图6-10 中断现场保护示意图

3) 转入相应的设备处理程序。cpu确认中断信号源，向它发送确认信号，让设备取消中断请求信号，之后把中断程序地址装入程序计数器
4) 中断处理。cpu从程序计数器找到中断程序，从设备控制器读设备状态，判断正常完成中断/异常结束中断，在内存和设备间交换数据/处理异常
5) 恢复cpu线程并退出中断
   (1) 若是屏蔽中断方式，就回到被中断的进程，从中断栈读出现场
   (2) 若是嵌套中断，会处理优先级更高的中断请求
6)

图6-11 中断处理流程

3. 除了4）外都是相同的，所以unix把1235集中起来，形成中断总控程序

-
- 设备驱动程序

一. 设备驱动程序概述

1. 设备驱动程序的功能
   1) 接受设备无关软件发来的命令、参数，转换成设备相关操作序列
   2) 检查请求合法性、了解设备工作状态、传递参数、设置设备工作方式
   3) 发出IO命令，根据设备空闲与否，立即启动或挂在设备队列
   4) 及时响应设备控制器的中断请求，调用对应中断处理程序
2. 设备驱动程序的特点
   1) 是低级的系统例程
   2) 是负责在设备无关软件和设备控制器之间通信和转换的程序
   3) 与设备控制器及IO硬件特效紧密相关，不同设备应有不同驱动
   4) 与IO设备的IO控制方式紧密相关，主要有中断驱动和DMA两种
   5) 一部分必须有汇编写，以做到与硬件紧密相关，有的还固化在ROM中
   6) 可重入，可能会在一次调用完成前再次被调用
3. 设备处理方式
   1) 为每类设备设一个进程，执行其IO，适用于大系统，如交互终端，打印机各一个
   2) 整个系统只有一个IO进程，或IO各一个
   3) 不专门设置设备处理进程，直接给用户或系统调用驱动，目前较常用

二. 设备驱动程序的处理过程

1. 将抽象要求转换为具体命令、参数。如盘块号转换成盘面、磁道号、扇区，只有驱动能做到
2. 校验服务请求是否合法，并终止进程或通知有错。如从打印机读数据，修改只读文件

3. 检查设备状态，指启动设备前，测试状态寄存器各状态，可能使进程等待

1)

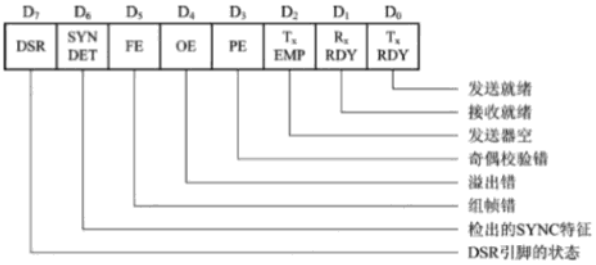


图6-12 状态寄存器中的格式

4. 向各寄存器传送命令、参数。如控制命令、传送速率、字符长度
5. 启动IO设备，向控制器的命令寄存器传控制命令
    1) 多道程序中，驱动程序一旦发出IO命令，启动操作后，驱动就把控制返回给IO系统，阻塞自己，到下个中断来时再被唤醒，IO操作是在设备控制器的控制下进行的，此时处理机可以并行干其他事

三. 对IO设备的控制方式

1. **轮询的可编程IO**方式：cpu向控制器发出IO指令时，把状态寄存器的忙闲标志busy置1，之后不断循环测试busy，是0时表示操作完成了
    1) 绝大部分时间在轮询测试，浪费cpu
2. **使用中断的可编程IO**方式：cpu发出IO命令后立即继续执行其他任务，由控制器完成指令后用控制线发送中断信号，cpu确认是正常完成中断后再处理
    1) cpu与io并行，利用率一般成百倍以上的提升

2)

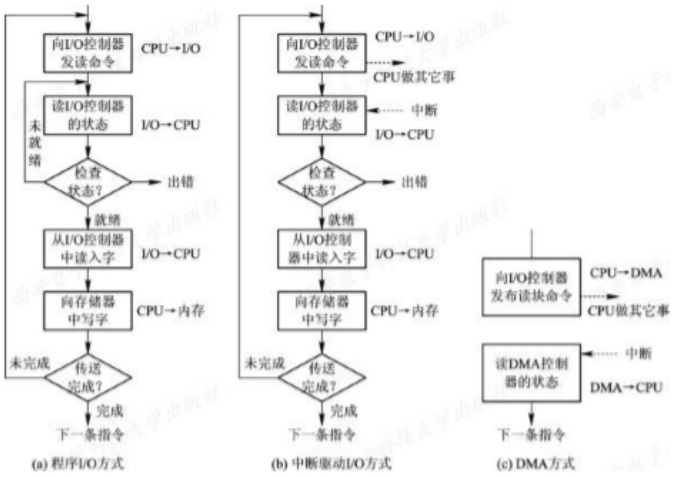


图6-13 程序I/O和中断驱动方式的流程

3. **直接存储器访问**方式(Direct Memory Access)
    1) 直接存储器访问方式的引入
        (1) 数据传输基本单位是数据块，每次至少传一个数据块
        (2) 传输数据时直接在**设备和内存间交换**的
        (3) 仅在传送数据块的开始和结束时才需cpu干预，由DMA控制传输
    2) DMA控制器的组成：与主机的接口、与块设备的接口、IO逻辑

(1)

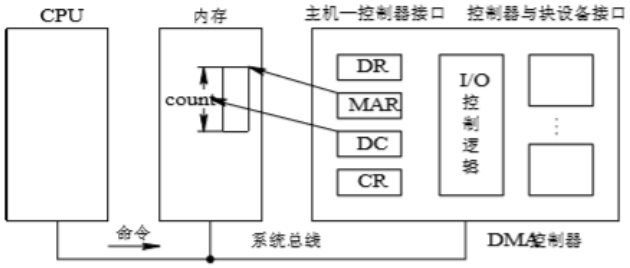


图 5-8 DMA 控制器的组成

(2) 命令/状态寄存器CR：cpu发来的IO命令、控制信息、设备状态

(3) 内存地址寄存器MAR：输入的内存起址，或输出的设备源址

(4) 数据寄存器DR：暂存待交换的数据

(5) 数据计数器DC：存放要操作的字数

3) DMA工作过程

(1) cpu向磁盘控制器发送命令，命令存进CR，地址存进MAR，字数存进DC，磁盘源址送进IO逻辑，启动DMA，数据送入DR，传到MAR指向的单元，--DC的内容，直至为0后发出中断请求

(2)

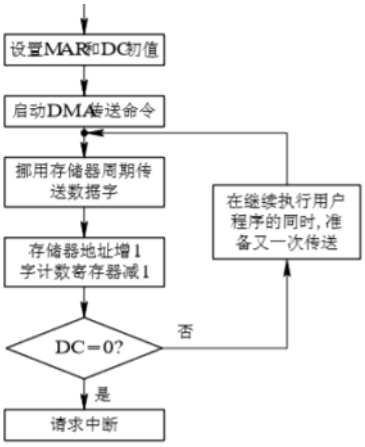


图 5-9 DMA 方式的工作流程图

4. **IO通道控制**方式

1) 通道方式的引入：DMA的发展，读写单位由一个数据块变成一组数据块

2) 通道程序：由一系列通道指令/命令构成，每条都包含：操作码（读/写/控制）、内存首地址、字数、通道程序结束位P（是不是通道最后一条指令）、记录结束标志（是不是一段指令的最后一条）

(1)

操　作	P	R	计　数	内存地址
WRITE	0	0	80	813
WRITE	0	0	140	1034
WRITE	0	1	60	5830
WRITE	0	1	300	2000
WRITE	0	0	250	1650
WRITE	1	1	250	2720

◆

◆ 与设备无关的IO软件

一. 与设备无关软件的基本概念(Device Independence)

1) 设备独立性/设备无关性：应用程序独立于具体使用的物理设备

2) 为提高 OS 的可适应性和可扩展性，在现代 OS 中都毫无例外地实现了它

1. 以物理设备名使用设备：早期OS必须用物理名称来控制设备

1）不灵活，不利于提高IO设备利用率

2. 引入逻辑设备名，实现IO重定向：不必更改程序，也可更换设备

3. 逻辑设备名到物理设备名的转换：程序执行时，需把逻辑地址转换为物理地址

二. 与设备无关的IO软件

1. 设备驱动程序的同一接口：方便添加新设备驱动程序，将抽象设备名映射到合适的驱动程序，保护设备，防止无权用户的访问

2. 缓冲管理：配置字符设备、块设备的缓冲

3. 差错控制：暂时性错误如电源波动等，用重试操作来纠正；持久错误如掉电，磁盘划痕、除以零等。磁盘盘块遭破坏只需记录其块号，以后不再使用即可

4. 对独立设备的分配与回收

1）确认空闲再分配，否则阻塞它，等被释放了再唤醒

5. 独立于设备的逻辑数据块：隐藏不同设备的不同数据交换单位，向高层软件提供统一大小的逻辑数据块

三. 设备分配

1. 设备分配中的数据结构

1）设备控制表DCT

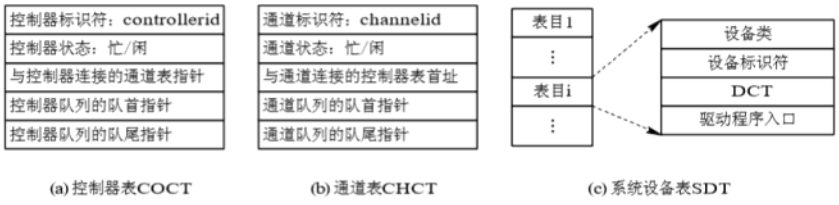
(1)

图 5-20 设备控制表

(2) 表项包含：设备类型、设备标识、忙闲状态、控制器表指针、重复执行次数（达到指定大小后认为传送失败）、请求PCB队列指针

2）控制器控制表(COCT)、通道控制表(CHCT)和系统设备表(SDT)

(1)



(a) 控制器表COCT (b) 通道表CHCT (c) 系统设备表SDT

图 5-21 COCT、CHCT 和 SDT

2. 设备分配时应考虑的因素

1）设备的固有属性：独占设备：直至进程释放前都由当前进程独占、共享设备：对访问先后次序合理调度、虚拟设备：同共享设备

2）设备分配算法：先来先服务、高优先级优先

3）设备分配中的安全性

(1) 安全分配方式：发出 I/O 请求后，便进入阻塞状态，直到其 I/O 操作完成时才被唤醒。摒弃了造成死锁的的“请求和保持”条件。缺点是进程进展缓慢，即 CPU 与 I/O 设备是串行工作的

(2) 不安全分配方式：发出 I/O 请求后仍继续运行，需要时又发出新请求，仅当进程所请求的设备已被另一进程占用时，才进入阻塞状态。 这种分配方式的优点是，一个进程可同时操作多个设备，推进迅速。缺点是可能造成死锁。因此，还应进行安全性计算

3. 独占设备的分配程序

1) 基本的设备分配程序

(1) 分配设备：根据 I/O 请求中的物理设备名，查找系统设备表(SDT)，从中找出其 DCT，若 DCT 中的设备状态字段为忙，或不通过安全性计算，便将 PCB 挂在设备队列上；安全才将设备分配给请求进程

(2) 分配控制器：把设备分配给请求 I/O 的进程后，再到其 DCT 中找出与该设备连接的控制器的 COCT，根据忙闲字段，将 PCB 挂在该控制器的等待队列上或将该控制器分配给进程

(3) 分配通道：在 COCT 中又可找到与该控制器连接的通道的 CHCT，再根据 CHCT 的忙闲字段，将 PCB 挂在该通道的等待队列上，或将该通道分配给进程

(4) 只有在设备、 控制器和通道三者都分配成功时，这次的设备分配才算成功。 然后，便可启动该 I/O 设备进行数据传送

2) 设备分配程序的改进

i. 上述程序不具备设备无关性：以物理设备名来提出 I/O 请求；采用单通路的 I/O 系统结构，容易产生“瓶颈”现象。为此，应从以下两方面改进，提高灵活性和分配的成功率

(1) 增加设备的独立性：用逻辑设备名请求 I/O。系统先从 SDT 中找出空闲的该类设备的 DCT，仅当所有该类设备都忙时，才把进程挂在其的等待队列上

(2) 考虑多通路情况：仅当所有的控制器(通道)都忙时，此次的控制器(通道)分配才算失败，才把进程挂在控制器(通道)的等待队列上

四. 逻辑设备名到物理设备名映射的实现

1. 逻辑设备表(Logical Unit Table)

1) 每个表目中包含三项： 逻辑设备名、物理设备名和设备驱动程序的地址

2) 当系统为进程分配设备时，将在 LUT 上建立一个表目

3) 以后再有进程请求该逻辑设备名时，系统便可查找 LUT，找到物理设备

2. LUT的设置问题

1) 只设一张：不允许重复逻辑设备名，只适合单用户系统，左图

2) 每个用户设一张：每个PCB设一张，右图

3)

逻辑设备名	物理设备名	驱动程序入口地址
/dev/tty	3	1024
/dev/printe	5	2046
⋮	⋮	⋮

(a)

逻辑设备名	系统设备表指针
/dev/tty	3
/dev/printe	5
⋮	

(b)

图 5-19　逻辑设备表

i.

ii.

iii.

iv.

v.

vi.
vii.
viii.
ix.
x.
xi. ---------我是底线-------

6用户层、缓冲、调度

2018年12月24日 9:10

-
- 用户层的IO软件

一. 系统调用与库函数

1. 系统调用：OS在用户层引进的中介过程，应用程序可通过它间接调用IO过程
   1) OS捕获到系统调用后，将CPU切换到核心态，转换到相应过程，完成IO，再切换回用户态，继续运行应用程序
   2) 早期，系统调用是应用程序取得OS服务的唯一途径
   3) 早期的系统调用以汇编语言提供，只有汇编语言的程序可以调用
2. 库函数：高级语言和新操作系统如C语和UNIX中，系统调用一一对应的库函数
   1) 库函数与调用程序连接在一起，嵌入在运行时装入内存的二进制程序中
   2) 内核提供OS基本功能，库函数扩展了OS内核，使用户方便取得服务
   3) 微软也定义了一套Application Program Interface，不过与系统调用并不一一对应
   4) 许多系统调用本身就采用C语言编写，以函数形式提供，可在C语言编写的程序直接调用
   5)

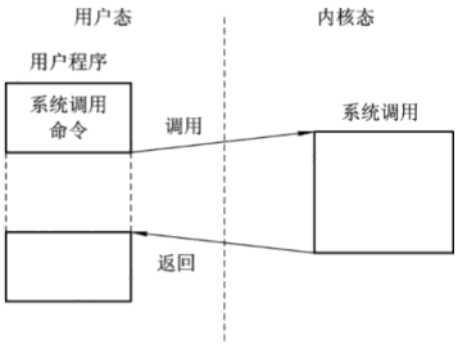
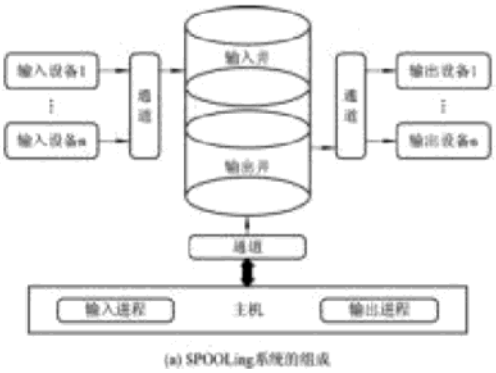


图6-20 系统调用的执行过程

二. 假脱机Spooling系统

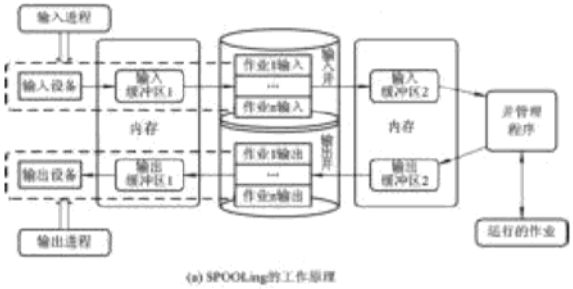
1) 20世纪50年代的脱机IO技术：用专门的外围控制机在低速IO设备和高速磁盘间传送数据

1. 多道程序系统中，可用两道程序模拟外围控制机的输入、输出，将一台物理IO设备虚拟为多台逻辑IO设备，把这种在联机情况下实现的**同时外围**操作称为SPOOLing(Simultaneaus Periphernal Operating On Line)，或称为假脱机
2. SPOOLing的组成
   1)

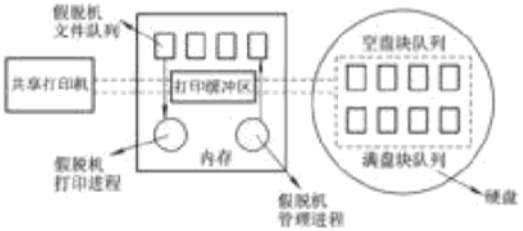


(a) SPOOLing系统的组成

   2) 输入/出井：**磁盘上的两个区域**，专门用于模拟脱机时的磁盘。用于以文件队列形式收容输入/出数据，称这些文件为井文件
   3) 输入/出缓冲区：内存中的两个缓冲区，用于缓和cpu和磁盘速度不匹配的矛盾

4) 预输入/缓输出进程：模拟外围控制机，（借助缓冲区）在设备和井间传数据
5) 井管理程序：控制作业与井之间的信息交换，作业提出请求时，由操作系统调用井管理程序，（借助另一个缓冲区）控制井内信息IO
6) 

(a) SPOOLing的工作原理

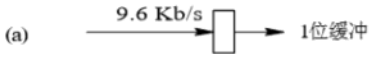
3. SPOOLing的特点
   1) 提高IO速度，**从操作低速IO设备变为操作磁盘**
   2) 将**独占设备改造成共享设备**，因为并没有把设备分配给进程，但磁盘中的空闲盘块和IO请求表是共享的
   3) 实现**虚拟设备**，每个进程都**以为自己独占了设备**
4. 假脱机打印机系统
   1) 打印机是常见独占设备，共享打印广泛用于多用户系统和局域网
   2) 

(c) 假脱机打印机系统的组成
   3) 假脱机管理进程：在磁盘缓冲区申请空闲盘块，暂存打印数据、申请打印请求表，填入打印要求，挂入井文件队列
   4) 假脱机打印进程：当打印机空闲，按井文件队首的打印请求表，将输出井的数据传送到内存缓冲区，交给打印机打印。若无打印请求就自我阻塞
   5) 打印操作是将数据送进缓冲区，在打印机空闲且排到队首时，cpu才用一个时间片进行打印。这些过程用户不可见
5. 守护进程daemon
   1) 用于执行一部分假脱机管理程序的功能，如申请空闲盘块，送入打印数据，返回盘块首址给进程，接收进程提交的完整打印请求等
   2) 是唯一允许使用守护的独占设备的进程，其他进程只能写请求文件
   3) 平常睡眠，出现新的井文件时被唤醒
   4) 除打印机外，服务器网络等也各可配置一个守护进程和井文件队列

- ◆
- ◆ 缓冲区管理

一. 缓冲的引入

1. 缓和 CPU 与 I/O 设备间速度不匹配的矛盾
2. 提高 CPU 和 I/O 设备之间的并行性
3. 减少对 CPU 的中断频率，放宽 CPU 中断响应时间的限制
   1) 如设置一个8位缓冲移位寄存器，即可将cpu中断响应时间放宽到8倍



2)

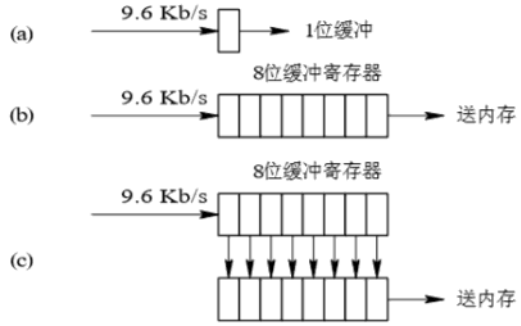
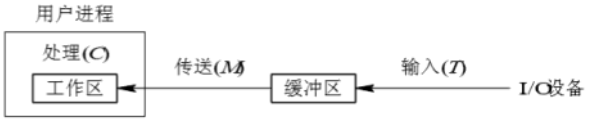


图 5-10　利用缓冲寄存器实现缓冲

4. 解决数据粒度不匹配问题
    1) 数据粒度：数据单元大小。越细化，粒度越小
    2) 粒度小的数据可以存一定量再操作，粒度大的数据可以分几次操作

二. 单缓冲区和双缓冲区

1. 单缓冲(Single Buffer)：进程每发出一 I/O 请求，系统便在主存分配一缓冲区
    1) 块设备输入时，假定从磁盘把一块数据输入到缓冲区的时间 为 T，数据传送到用户区的时间为 M，而 CPU 对这一块数据处理(计算)的时间为 C。由于 T 和 C 可并行，当 T>C 时，系统对每一块数据的处理时间为 M+T，反之则为 M+C，故可把系统对每一块数据的处理时间表示为 Max(C，T)+M



2)

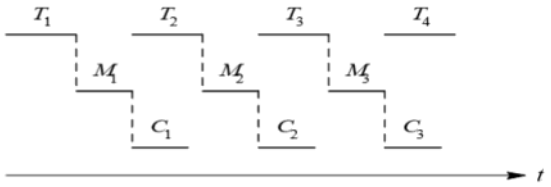
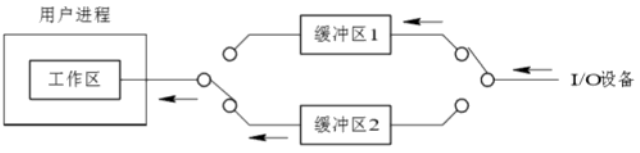


图 5-11　单缓冲工作示意图

    3) 新一行数据输入时进程被挂起；旧一行数据输出时进程被阻塞
2. 双缓冲(Double Buffer) /缓冲对换(Buffer Swapping)
    1) 第一缓冲区数据传送时，不用等待，转而使用第二缓冲区即可
    2) 处理一块数据的时间为 Max{C+M，T}。如果 C+M<T，可使块设备连续输入；如果 C+M>T，则可使 CPU 不必等待设备输入
    3) 处理n块数据的时间为 T+Max{C+M，T}+C



4)

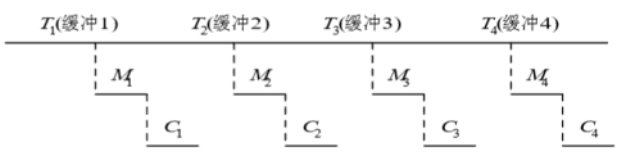


图 5-12　双缓冲工作示意图

    5) 双向数据传输同理，可以双方及其各设两个缓冲区

6)

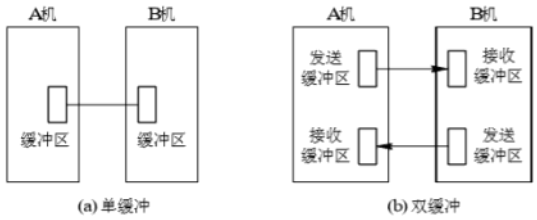


图 5-13　双机通信时缓冲区的设置

三. 环形缓冲区

1) 输入输出时间相差较大时双缓冲仍不够理想，需要多缓冲

1. 环形缓冲区的组成

1) 等大的多个缓冲区：空缓冲区 R、装满数据的缓冲区 G、计算进程正在使用的现行工作缓冲区 C

2) 多个指针：计算进程下个可用缓冲区 G 的指针 Nextg、输入进程下次可用的空缓冲区 R 的指针 Nexti、计算进程正在使用的缓冲区 C 的指针 Current

3)

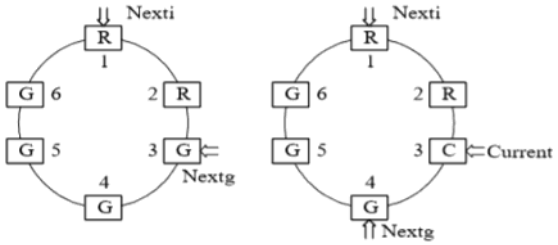


图 5-14　循环缓冲

2. 环形缓冲区的使用

1) Getbuf过程：

(1) 计算：将 Nextg 所指示的 R 提供给进程使用，改为C，并令 Current 指针指向其第一个单元，同时将 Nextg 移向下一个 G

(2) 输入：将指针 Nexti 所指示的 R 提供给输入进程，同时将 Nexti 指针移向下一个 R

2) Releasebuf过程：

(1) 计算：提取完毕 C 时调用，将 C 释放，改为 R

(2) 输入：装满 R 时调用，将 R 释放，改为 G

3. 进程间的同步问题

1) Nexti 指针追赶上 Nextg 指针：系统受计算限制

(1) 即输入过快，无空区，应阻塞输入进程直至计算进程调用 Releasebuf 过程

2) Nextg 指针追赶上 Nexti 指针：系统受 I/O 限制

(1) 即计算过快，无满区，应阻塞计算进程直至输入进程调用Releasebuf过程

四. 缓冲池Buffer Pool

1) 是即可输入又可输出的公用缓冲池

2) 缓冲区仅是内存块链表，缓冲池是数据结构+操作函数

3) 缓冲池的每个缓存区都有首部和缓冲体两部分

(1) 首部包含缓冲区号、设备号、数据块号、同步信号量、队列指针

1. 缓冲池的组成：空缓冲区；装满输入数据的缓冲区；装满输出数据的缓冲区；用于收容/提取 输入/输出数据的工作缓冲区

1) 空缓冲队列 emq。其队首指针 F(emq)和队尾指针 L(emq)分别指向其首缓冲区和尾缓冲区

2) 输入队列 inq。其队首指针 F(inq)和队尾指针 L(inq)分别指向其首缓冲区和尾缓冲区

3) 输出队列 outq。其队首指针 F(outq)和队尾指针 L(outq)分别指向其首缓冲区和尾缓冲区

2. Getbuf 过程和 Putbuf 过程

1) Addbuf(type,number)过程。将number所指示的缓冲区B挂在type 队列上
2) Takebuf(type)过程。从 type 所指示的队列的队首摘下一个缓冲区
3) 缓冲池中的队列本身是临界资源，访问队列，既应互斥，又须同步，可为每队列设置一个互斥信号量MS(type)。 此外，为了保证诸进程同步地使用缓冲区，又为每个缓冲队列设置了一个资源信号量RS(type)

```
(1) Procedure Getbuf(type){
        Wait(RS(type));
        Wait(MS(type));
            B(number)=Takebuf(type);
        Signal(MS(type));}
(2) Procedure Putbuf(type,number){
        Wait(MS(type));
            Addbuf(type,number);
        Signal(MS(type));
        Signal(RS(type));}
```

3. 缓存区工作方式

1)

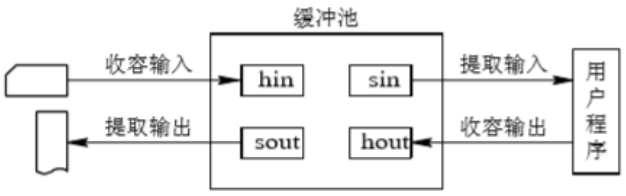


图 5-15 缓冲区的工作方式

2) 收容输入。输入进程输入数据时，调用 Getbuf(emq)过程，从空缓冲队列 emq 队首摘下一空缓冲区，作为收容输入工作缓冲区 hin。输入数据， 再调用 Putbuf(inq，hin)过程，将该缓冲区挂在输入队列 inq 上
3) 提取输入。计算进程需要数据时，调用 Getbuf(inq)过程，从输入队列 inq 队首取得一个缓冲区，作为提取输入工作缓冲区 sin 。计算进程用完该数据后，再调用 Putbuf(emq，sin)过程，将该缓冲区挂到空缓冲队列 emq 上
4) 收容输出。计算进程需要输出时，调用 Getbuf(emq)过程从空缓冲队列 emq 队首取得一空缓冲区，作为收容输出工作缓冲区 hout。装满输出数据后，又调用 Putbuf(outq，hout)过程，将该缓冲区挂在 outq 末尾
5) 提取输出。由输出进程调用 Getbuf(outq)过程，从输出队列的队首取得一装满输出数据的缓冲区，作为提取输出工作缓冲区 sout。在数据提取完后，再调用 Putbuf(emq，sout) 过程，将该缓冲区挂在空缓冲队列末尾

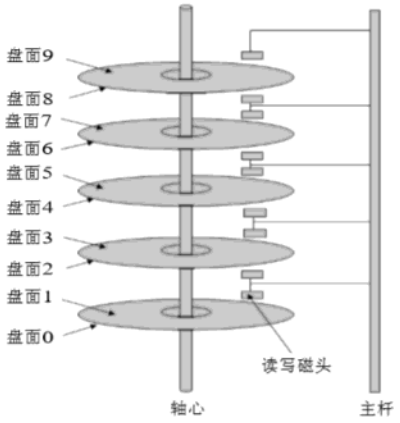
-
- 磁盘存储器的性能和调度

1. 提高磁盘性能除了考改善寻道调度算法外还可靠提高IO速度、冗余技术

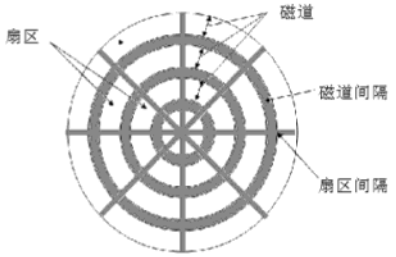
一. 磁盘性能简述

1. 数据的组织和格式
   1) 磁盘设备包括一或多个物理盘片，每片分一或两个存储面(surface)
   2) 每面被组织成若干同心环，称为柱面/磁道(track)，各磁道间留有间隙gap
   3) 每条磁道上可存储同数目的二进制位，内层磁道的磁盘位数密度更高
   4) 每条磁道逻辑上划分成若干个扇区(sectors)，软盘 8 ~ 32 个，硬盘数百
   5) **扇区又称盘块**(或数据块)。各扇区间保留一定扇区间隙gap

(1)



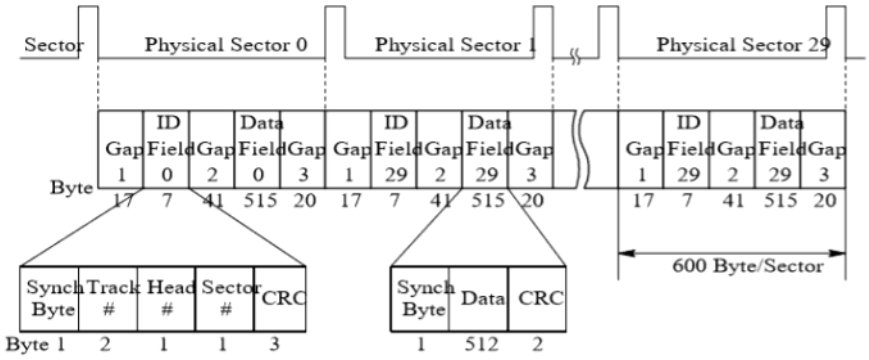
(a) 磁盘驱动器的结构



(b) 磁盘的数据布局

6) 现代磁盘常把盘面划分成若干环带，外层环带拥有更多扇区
7) 大多数磁盘都隐藏了细节，只向系统提供虚拟几何的磁盘规格
8) 磁盘存储数据前需要先低级格式化，如温切斯特盘，每磁道30扇区
   (1) 标识符字段：每个扇区的其中88字节。其中一个字节的 SYNCH 具有特定的位图像，作为该字段的定界符， 利用磁道号、 磁头号及扇区号三者来标识一个扇区；CRC 字段用于段校验
   (2) 数据字段：每个扇区的剩余 512 个字节，用于存储数据

(3)



9) 磁盘低级格式化后即可分区，每个分区是逻辑上独立的盘，每个分区的起始扇区都记录在磁盘0扇区的分区表，该表中有一个被标记成活动的主引导记录分区，保证从硬盘能引导系统
10) 真正使用磁盘前还需再高级格式化一次，设置引导块、空闲存储管理、根目录、空文件系统，再在分区表标记其文件系统

2. 磁盘的类型
   1) 硬盘、软盘；单片盘、多片盘；固定头、活动头/移动头磁盘
   2) 固定头磁盘：每条磁道上都有一读/写磁头，所有的磁头都被装在一刚性磁臂中，有效地提高了磁盘的 I/O 速度。主要用于大容量磁盘上
   3) 移动头磁盘：每个盘面仅配有一个磁头，也被装入磁臂中。为能访问该盘面上的所有磁道，该磁头必须能移动寻道，仅能以串行方式读/写，致使 I/O 速度较慢； 由于其结构简单，仍广泛应用于中小型磁盘设备中。在微型机上配置的温盘和软盘都采用移动磁头结构
3. 磁盘访问时间
   1) 寻道时间 Ts ： 磁臂(磁头)移动到指定磁道上所经历的时间
      (1) 是启动磁臂的时间 s 与磁头移动 n 条磁道所花费的时间之和
         i. $T_s = m \times n + s$
         ii. m与磁盘驱动器速度有关，一般磁盘m=0.2；高速磁盘m<=0.1
         iii. s=2ms，一般温盘的寻道时间约5~30ms
   2) 旋转延迟时间 Tr ： 指定扇区移动到磁头下面所经历的时间

(1) 不同的磁盘类型中，旋转速度至少相差一个数量级，如软盘为300 r/min，硬盘一般为7200～15 000 r/min，甚至更高

(2) 硬盘，每转需时4 ms，平均旋转延迟时间Tr为2 ms；而软盘平均Tr为50～100 ms

3) 传输时间Tt：把数据从磁盘读出或向磁盘写入数据所经历的时间

(1) 与读/写字节数b和旋转速度有关

(2) $$T_t = \frac{b}{rN}$$

(3) r为磁盘每秒钟的转数；N为一条磁道上的字节数

4) 当一次读/写的字节数相当于半条磁道上的字节数时，Tt与Tr相同，此时

(1) $$T_a = T_s + \frac{1}{2r} + \frac{b}{rN}$$

(2) 寻道时间和旋转延迟时间基本上与所读/写数据多少无关，且通常占据了访问时间中的大头

(3) 现传输速率已高达80m/s，提高传输效率最快的方式已是适当地集中数据传输了

二. 早期的磁盘调度算法

1. 先来先服务(First Come First Served)

1) 公平、简单、不会使某进程长期得不到满足；平均寻道时间长

2) 平均寻道距离大，只适于磁盘IO进程数少的场合

2. **最短寻道时间优先**(Shortest Seek Time First)

1) 即选择最近的磁道。仍不能保证平均寻道时间最短，仅优于fcfs

(从100号磁道开始)	
被访问的下一个磁道号	移动距离(磁道数)
55	45
58	3
39	19
18	21
90	72
160	70
150	10
38	112
184	146
平均寻道长度: 55.3	

图 5-25 FCFS 调度算法

(从100号磁道开始)	
被访问的下一个磁道号	移动距离(磁道数)
90	10
58	32
55	3
39	16
38	1
18	20
150	132
160	10
184	24
平均寻道长度: 27.5	

图 5-26 SSTF 调度算法

三. 基于扫描的磁盘调度算法

1) 进程“饥饿”(Starvation)现象：新请求不断到达，低优先级进程永不被满足

1. **扫描**(SCAN)算法/电梯调度算法

1) SCAN 算法优先考虑磁头当前移动方向，正向无磁道需访问时才反向移动

2. **循环扫描**(CSCAN)算法

1) 扫描算法不能照顾到磁头换向后新到的进程

2) 循环扫描使磁头移到最外道后马上回到最里

3) T为扫描一轮的时间，Smax为移动的时间，则新到进程等待时间由2T变为T+Smax

4)

(从 100#磁道开始，向磁道号增加方向访问)	
被访问的下 一个磁道号	移动距离 (磁道数)
150	50
160	10
184	24
90	94
58	32
55	3
39	16
38	1
18	20
平均寻道长度：27.8	

图 5-27　SCAN 调度算法示例

(从 100#磁道开始，向磁道号增加方向访问)	
被访问的下 一个磁道号	移动距离 (磁道数)
150	50
160	10
184	24
18	166
38	20
39	1
55	16
58	3
90	32
平均寻道长度: 35.8	

图 5-28　CSCAN 调度算法示例

3. NStepSCAN 和 FSCAN 调度算法
   1) NStepSCAN
      (1) “磁臂粘着” (Armstickiness)：进程反复请求对某磁道的 I/O 操作，垄断了整个磁盘设备，磁臂停留在某处不动
      (2) N步扫描将磁盘请求队列分为若干长为N的子队列，FCFS处理
      (3) 通过把新进程的请求放在其他子队列，避免粘着
      (4) n=1时退化为fcfs，n很大时接近scan
   2) FSCAN
      (1) N步扫描的退化，只分成当前子队列和新到子队列
          i.
          ii.
          iii.
          iv.
          v.
          vi.
          vii.
          viii.
          ix. ---------我是底线-------

7文件系统、逻辑结构

2018年12月19日 13:09

-
- 文件和文件系统

1. 内存是易失性设备，断电后信息会丢失，而其容量十分有限，所以需要外存
2. 外存文件系统的管理功能是将程序和数据组织为文件的方式实现的

一. 数据项、记录、文件

1. 数据项：最低级的数据组织形式
   1) 基本数据项/字段：描述某对象某属性的字符集
      (1) 基本数据项除了数据名外，还应有数据类型
      (2) 名和类型共同定义了数据项的“型”，数据称为“值”
      (3) 是数据组织中可命名的最小逻辑数据单位。如，学号、姓名、年龄等
   2) 组合数据项/组项：若干个基本数据项组成的
      (1) 如，工资是组项，由基本工资、工龄工资和奖励工资等基本项组成
2. 记录：一组数据项的集合，描述对象某方面的属性
   1) 具体要包含哪些数据项，取决于要描述哪个方面
   2) 关键字key：若干个能标识一条记录的数据项的集合
   3)

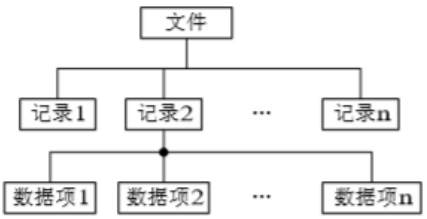


图 6-1 文件、记录和数据项之间的层次关系

3. 文件：创建者定义的、具有文件名的一组相关元素的集合
   (1) 有结构文件/记录式文件：若干相关记录组成
   (2) 无结构文件/流文件：字符流
   1) 是文件系统中最大的数据单位，描述了一个对象集
   2) 属性
      (1) 文件类型：如源文件、目标文件、可执行文件
      (2) 文件长度：单位是字节、字、块的长度
      (3) 物理位置：指示文件所在设备，及设备内的地址
      (4) 建立时间：最后一次修改的时间

二. 文件名和类型

1. 文件名和扩展名
   1) 文件名。空格常不能作为文件名，因为它常作为分隔命令、参数、数据项的分隔符。NTFS很好的支持长文件名。UNIX和Linux都区分大小写
   2) 扩展名/后缀名：文件名后的若干个附加字符
      (1) 大多数系统是用.分开，长度一般是1~4个字符
2. 文件类型
   1) 按用途分类
      (1) 系统文件：由系统软件构成
         i. 一般只许用户调用，不许用户读、改，甚至不直接对用户开放

(2) 用户文件：由用户的源代码、目标文件、可执行文件或数据等构成

i. 用户将这些文件委托给系统保管

(3) 库文件：由标准子例程及常用的例程等所构成的文件

i. 允许用户调用，但不许修改

2) 按文件中数据的形式分类

(1) 源文件：由源程序和数据构成的文件

i. 通常由终端或输入设备输入的源程序和数据所形成的文件都属于源文件

ii. 通常是由 ASCII 码或汉字所组成的

(2) 目标文件：源程序经编译过，尚未链接的目标代码所构成

i. 属于二进制文件

(3) 可执行文件：编译产生的目标代码再链接后所形成

3) 按存取控制属性分类

(1) 只执行文件：只允许被核准的用户调用执行，既不许读，更不许写

(2) 只读文件：只允许文件主及被核准的用户去读，但不许写

(3) 读写文件：允许文件主和被核准的用户去读或写的文件

4) 按组织形式和处理方式分类

(1) 普通文件：由 ASCII 码或二进制码组成的字符文件

(2) 目录文件：由文件目录组成的，用来管理和实现文件系统功能的系统文件，通过目录文件可以对其它文件的信息进行检索

(3) 特殊文件：特指系统中的各类 I/O 设备。为方便管理，所有设备都视为文件，以文件方式提供给用户使用

三. 文件系统的层次结构

1)

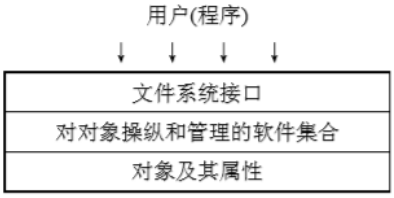


图 6-2 文件系统模型

1. 对象及其属性

1) 文件：直接管理对象

2) 目录：方便存取和检索而配置的，提高存取速度的关键

3) 磁盘/磁带存储空间：有效管理以提高外存利用率，提高存储速度

(1) 含有文件名、属性、物理地址

2. 对对象操纵和管理的软件集合

1) 是文件管理系统的核心部分，包括:

(1) 管理文件存储空间、管理文件目录、逻辑/物理地址转换机制、 文件读/写管理，文件的共享与保护等

(2) 可分为：

i. IO控制层/设备驱动程序层：最低层，由磁盘驱动程序构成

ii. 基本文件系统层：交换内存与磁盘间的数据块

iii. 基本IO管理程序：逻辑/物理块号转换，管理空闲盘块，指定IO缓冲

iv. 逻辑文件系统：用符号文件名访问文件、保护文件

3. 文件系统的接口
    1) 命令接口：用户与文件系统交互的接口
    2) 程序接口：用户程序与文件系统的接口
        (1) 用户程序可通过系统调用来取得文件系统的服务

四. 文件操作

1. 最基本的文件操作
    1) 创建文件：分配必要的外存空间，建立一个目录项
    2) 删除文件：先找到目录项，使之成为空项，然后回收存储空间
    3) 读文件：找到目录项，得到在外存中的位置
    4) 写文件：给出文件名及内存中的 (源)地址，先找到目录项，利用写指针进 行写操作
    5) 截断文件：想更新文件，如果文件名及其属性均无改变时，将原有文件的长度设置为 0即可，或者说是放弃原有的文件内容
    6) 设置文件的读/写位置：设置文件读/写指针的位置，以便不从始端开始操作。使顺序存取变为随机存取
2. 文件的“打开”和“关闭”操作
    1) 打开(open)系统调用：将指名文件的属性(包括该文件在外存上的物理位置)从外存拷贝到内存打开文件表的一个表目中，并将该表目的编号/索引返回给用户（用户可根据索引，获得其信息）
    2) 关闭”(close)系统调用：把该文件从打开文件表中的表目上删除掉
3. 其它文件操作
    1) 最常用的是对文件属性进行操作，如改变已存文件的文件名、改变文件的拥有者(文件主)、改变对文件的访问权， 以及查询文件的状态(包括文件类型、大小和拥有者以及对文件的访问权等)
    2) 另一类是有关目录的，如创建/删除目录，改变当前目录和工作目录等
    3) 此外，还有实现文件共享和对文件系统进行操作的系统调用
4. 某些操作的组合实现新操作：如创建+写入，实现复制

-
- 文件的逻辑结构

1. 逻辑结构(File Logical Structure)/文件组织(File Organization)：用户观察到的文件组织形式，是用户可直接处理的数据及其结构，独立于文件物理特性
2. 文件的物理结构：文件的存储结构，指文件在外存上的存储组织形式。不仅与存储介质的存储性能有关，而且与所采用的外存分配方式有关
3. 两种结构都影响检索速度，本章介绍逻辑，物理见8章外存组织

一. 文件逻辑结构的类型

1. 按文件是否有结构分类
    1) 有结构文件/记录式文件
        (1) 定长记录：检索高效，删改方便，广泛用于数据处理
        (2) 变长记录：记录长度不同的文件，但各文件长度都可知，检索慢，不易删改，但适合并广泛用于商业领域
    2) 无结构文件/流失文件：长度以字节为单位，用读写指针访问下一字符，

可视作记录长固定1字节的记录式文件。如源码、可执行文件、库函数

2. 有结构文件按组织方式分类
    1) 顺序文件：按某种顺序排列一系列记录的文件
    2) 索引文件：为每个可变长记录文件设一张索引表，每个表项对应一记录
    3) 索引顺序文件：为记录分组，每组记录的首条填入索引表

二. 顺序文件(Sequential File)

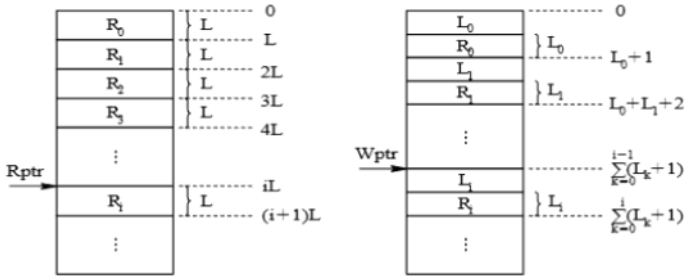
1. 顺序文件的排列方式
    1) 串结构：按存入时间排序。必须从头检索
    2) 顺序结构：以某有唯一性的字段作为关键字。可二分、插值、跳步查找

2. 优缺点：顺序文件适合批量存取；难以增删，一般用事务/log文件辅助增改

三. 记录寻址

1. 隐式寻址方式：靠读指针Rptr保存下条记录的首址，读完当前记录后+=当前记录长度L；同理读指针Wptr。变长记录文件随机访问需要读前i-1个记录的长度

1)



(a) 定长记录文件　　(b) 变长记录文件

图 6-3　定长和变长记录文件

2. 显式寻址方式：对定长记录的随机访问
    1) 利用记录位置（给记录编号i）

$A_i = i \times L$	$A_i = \sum_{i=0}^{i-1} L_i + i$
定长文件	变长文件

    2) 利用唯一关键字，逐个比较到匹配为止，如目录检索关键字是符号文件名
        (1) 商业领域中常用的是变长，关键字搜索

四. 索引文件(Index File)

1. 按关键字建立索引：为每条变长记录设一表项，存储其逻辑首址和长度L，按关键字排序，该索引表是定长记录顺序文件，可以二分查找，有及时性
2. 具有多个索引表的索引文件：为每种能成为检索条件的域（属性或关键字）各配一张索引表。将顺序查找改成随机查找；但增加了存储开销

3.

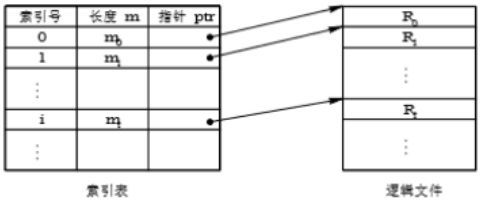


图 6-4　索引文件的组织

五. 索引顺序文件(Index Sequential File)

1. 索引顺序文件的特征：按关键字的顺序组织的前提下，引入索引表实现随机访问，再增加溢出overflow文件方便增删改

2. 一级索引顺序文件：记录分组，每组的首项关键字和首址填入索引表

1)

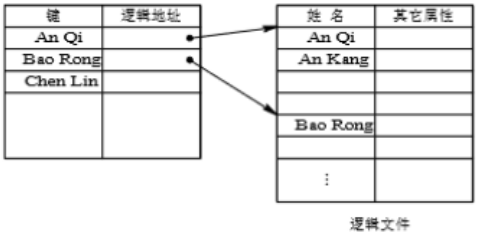


图 6-5 索引顺序文件

2) 先按关键字找索引表，再从首项顺序检索具体记录

3) 顺序文件平均查N/2次，索引顺序文件平均查√N个记录，提高√N /2倍

3. 两级索引顺序文件：为许多低级索引再建一张高级索引表，存低级索引首项。平均查(3/2)$_3$√N次。例100 0000个记录，一级查1000次，二级查150次

六. 直接文件和哈希文件

1. 直接文件：可以根据关键字段值直接找到物理地址的文件

1) 键值转换(Key to address transformation)：由关键字得到物理地址

2. 哈希Hash文件：能用Hash散列函数将键值K转换为相应记录的文件

1) 其实不是直接算出地址，而是算出表项指针A=H（K），再去查表

2) 一般把哈希函数作为标准函数存进系统，供存取文件时调用

3)

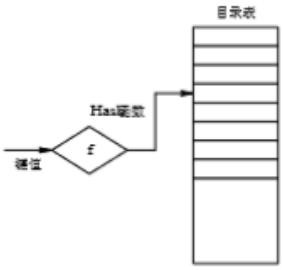


图 6-6 Hash 文件的逻辑结构

i.

ii.

iii.

iv.

v. ---------我是底线-------

7目录、共享、保护

2019年1月6日 18:30

-
- 文件目录

1. 目录管理的要求：
   1) 按名存取：最基本的功能，以后提供文件名，系统返回存储位置
   2) 高速检索：大中型文件系统的目标
   3) 文件共享：多用户系统的目标，一般通过提供共享副本实现
   4) 允许文件重名：指对不同用户可以对不同文件采用相同名

一. 文件控制块和索引结点

   1) 文件控制块的有序集合即为文件目录
   2) 文件目录也可视作文件：目录文件

1. 文件控制块FCB
   1) 基本信息类
      (1) 文件名：唯一标识
      (2) 文件物理位置：设备名、起始盘块号、占用盘块数或字节数
      (3) 文件逻辑结构：指示文件是否定长，是否有结构、记录数
      (4) 文件的物理结构，指示文件是顺序文件、链接式文件或索引文件
   2) 存取控制信息类：文件主、核准用户、一般用户的不同存取权限
   3) 使用信息类：建立、修改时间、使用信息（使用进程数，锁，是否修改）
   4) 

图 6-15 MS-DOS 的文件控制块

2. 索引结点
   1) 索引结点的引入：查找目录通常需要将存放目录的第一个盘块的目录调入内存，逐个比较，找不到还需调入下一盘块，平均需要总盘块数N+1/2次
      (1) 实际上检索时只有匹配的文件名及其物理地址是有用的信息
      (2) UNIX的文件描述信息放在索引结点i中
      (3) 

图 6-16 UNIX 的文件目录
   2) 磁盘索引结点
      (1) 文件主标识符：拥有该文件的个人或小组的标识符
      (2) 文件类型：正规文件、目录文件或特别文件
      (3) 文件存取权限：各类用户对该文件的存取权限
      (4) 文件物理地址：每个索引结点中有 13 个地址项 iaddr(0) ~ iaddr(12)，以直接或间接方式给出数据文件所在盘块的编号
      (5) 文件长度：以字节为单位的文件长度

(6) 文件连接计数：表明在本文件系统中所有指向该文件的指针计数

(7) 文件存取时间：本文件最近被进程存取、修改的时间、索引结点最近被修改的时间

3) 内存索引结点：文件打开后，磁盘索引结点拷贝到内存中，便于以后使用，增加了以下内容

(1) 索引结点编号：用于标识内存索引结点

(2) 状态：指示 i 结点是否上锁或被修改

(3) 访问计数：每当有一进程要访问此 i 结点时，++该访问计数，访问完再--

(4) 文件所属文件系统的逻辑设备号

(5) 链接指针：分别指向空闲链表和散列队列的指针

二. 简单的文件目录

1. 单级文件目录：整个文件系统只有一张目录表，每个文件占一项

1) 查找速度慢：评价需顺序查找N/2次

2) 不允许重名：新建文件时，检查不与其他文件名相同

3) 不便于共享：必须用唯一文件名来访问，只适于单用户环境

4)

文件名	物理地址	文件说明	状态位
文件名 1			
文件名 2			
…			

图 6-17 单级目录

2. 两级文件目录：为每个用户单独建一个用户文件目录 UFD(User File Directory)，再在系统中建一个主文件目录 MFD(Master File Directory)，每个用户占一行

(1)

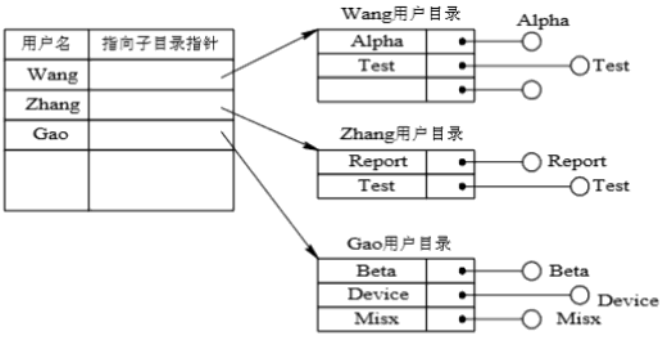


图 6-18 两级目录结构

1) 提高了检索目录的速度： n 个用户各最多为 m 个 目录项，则最多只需检索 n + m 个目录项。但如果是采用单级目录结构，则最多需检索 n × m 个目录项。检索效率提高约 n/2 倍

2) 在不同的用户目录中，可以使用相同的文件名，只有同用户同文件名才需要重命名

3) 不同用户可使用不同文件名访问同一共享文件。在各用户之间完全无关时，这种隔离是一个优点；但当多个用户间要合作，访问对方文件时，这种隔离便成为一个缺点，会使诸用户之间不便于共享文件

三. 树形结构目录Tree-Structured Directory

1. 树形目录：主目录为唯一的根目录，数据文件为树叶，其他子目录为结点

1)

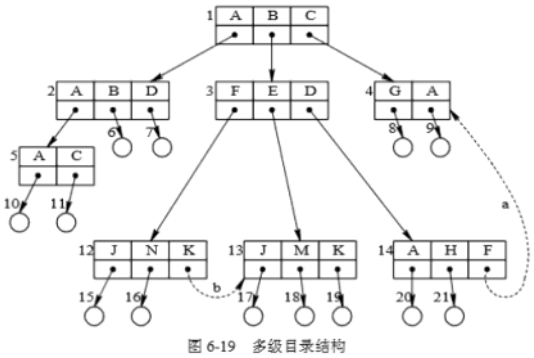


图 6-19　多级目录结构

2)　(圆圈为数据文件，方框为目录文件)

2. 路径名和当前目录
    1) 绝对路径名(absolute path name)：根目录到文件的通路间，每一结点前后都用 / 连接，即构成其唯一路径名，因为从根到任一树叶都只有唯一通路
    2) 当前目录(Current Directory) /工作目录：进程访问范围的父结点
    3) 相对路径名(relative path name)：从当前目录开始的路径名
    4) 查询速度快，层次结构清晰，文件管理保护有效，存取权限易区分
    5) 但磁盘访问次数多
3. 目录操作
    1) 创建目录：用户先创建UFD，再在其中新增不重名的子目录
    2) 删除目录：空目录可直接删，非空目录有两种处理方式：
        (1) 不允许删，只能递归清空其子目录才能删，如MS-DOS
        (2) 允许删，直接将全部子目录和文件清空，较危险
    3) 改变目录：即设置当前目录，默认是到主目录
    4) 移动目录：将文件或子目录在不同父文件间转换
    5) 链接link操作：让文件具有多个父目录，方便共享
    6) 查找：精确匹配文件或局部匹配文件

四. 目录查询技术

1. 线性检索法/顺序检索法：依次读入路径分量名，顺序比较文件名，按索引结点号插索引表，读入其盘块，直到找到文件，任一分量找不到都应返回错误信息

1)

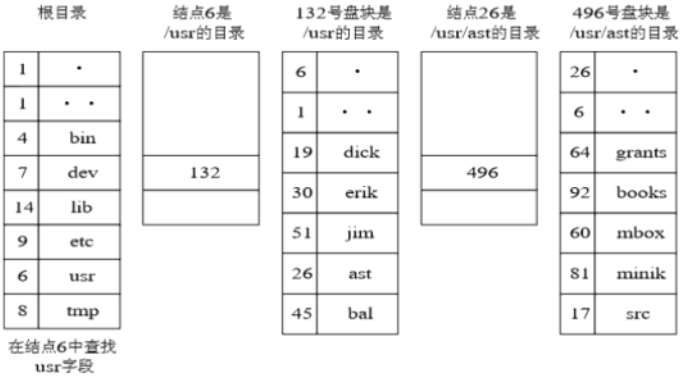


图 6-20　查找/usr/ast/mbox 的步骤

2. Hash方法：用hash函数求出文件名对应的索引值，再到hash索引目录找
    1) 不适用于使用了通配符的文件名
    2) 冲突：不同文件名可能转换为同一hash值
        (1) 如果hash值在索引表找到目录项为空，说明尚无指定文件
        (2) 若目录项的文件名匹配，说明查找成功

(3) 若非空但不匹配，说明是发生了冲突，因在hash值上加一与目录长度互质的常数，再重新查

-
- 文件共享

1) 可称为绕弯路法和连访法

一. 基于有向无循环图实现文件共享

1. 有向无循环图DAG(Directed Acyclic Graph)
   1) 不同用户各自的父目录指向同一文件，多用户以对称方式共享文件
   2) 会产生回路，破坏树形结构
   3) 增加新内容可能要增加新盘块，新增部分对其他用户不可见
   4)

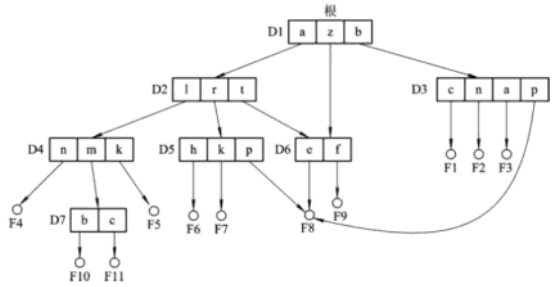


图7-13 有向无循环图目录层次

2. 利用索引结点：将物理地址等文件属性放在索引结点
   1) 文件目录只含文件名和索引结点号
   2) 新增信息时修改索引结点的内容，使其他用户可见修改
   3) 索引结点中应记录链接计数count，记录共享用户数
   4) count不为0就不能删除文件，计帐系统中，文件主可能要为此“付账”

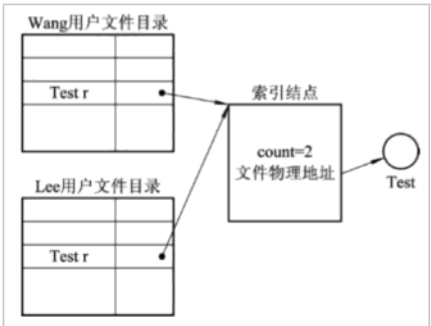


图7-14 基于索引结点的共享方式

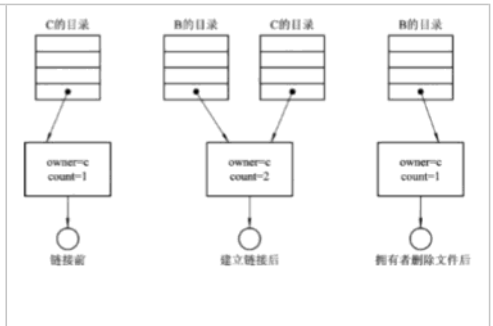


图7-15 进程B链接前后的情况

二. 利用符号链接（软链接）实现文件共享

1. 利用符号链接(Symbolic Linking)的基本思想：允许多个父目录存在，除了唯一主父目录外，都是链接父目录，靠符号链接。符号链接视作虚线，则实线部分仍是简单树，删查都很方便
   1)

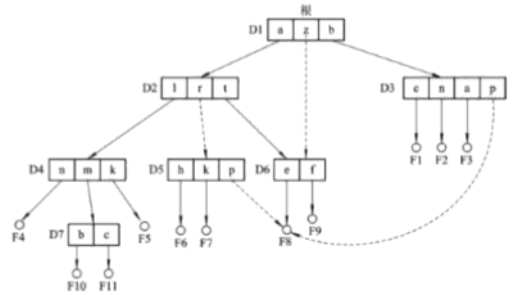


图7-16 使用符号链接的目录层次

2. 利用符号链实现共享：在链接父目录中创建link型文件，取名同文件名，内容为文件的路径名
   1) 即新建一个文件，里面写着该文件的目录，在linux中是ln -s建立的
3. 利用符号链实现共享的优点：
   1) 只有文件主拥有索引结点号，删除文件只会导致访问失败并自动删除链接，不会造成悬空指针误操作
   2) 适用于网络链接分布在世界各地的计算机的文件
4. 利用符号链的共享方式存在的问题：

1) 链接访问可能需要多次读盘，开销大
2) 链接本身也是文件，需要额外分配索引结点，浪费空间
3) 遍历traverse文件系统可能反复读到某一链接，使共享文件被反复拷贝

◆

◆ 文件保护

1. 影响文件安全的主要因素：
   1) 人为因素：人们有意无意的行为使数据被破坏或丢失
   2) 系统因素：系统异常导致数据破坏或丢失，如磁盘故障
   3) 自然因素：随时间推移，磁盘上的数据会逐渐消失
2. 为确保文件安全性可采取：
   1) 存取控制权限，防止人为因素
   2) 采取系统容错技术，防止系统因素
   3) 建立后备系统，

一. 保护域(Protection Domain)

1. 访问权(Access right)<对象名，权集>：进程对某对象执行操作的权力
2. 保护域/域：进程拥有某访问权的对象的集合，进程只能在指定域中执行操作

1)

图7-17 三个保护域

3. 进程和域间的静态联系：进程和域之间可以一一对应，即在进程的整个生命期中，可用资源是固定的，这种域称为“静态域”
   1) 进程运行全过程都受限于同一个域，会使赋予进程的访问权超过实际需要
4. 进程和域之间的动态联系方式：一个进程对应多个域，每个阶段不同域
   1) 称其为动态联系方式，需要系统中有保护域切换功能

二. 访问矩阵(Access Matrix)，

1. 访问矩阵：描述系统访问控制的矩阵，行为域，列为对象
   1) 访问权access(i, j)定义了在域Di中执行 的进程能对对象Qj所施加的操作集
2. 具有域切换权的访问矩阵
   1) 将切换能力作为一种权力S
   2) 当且仅当switch∈access(i, j)时，才允许进程从域i切换到域j

3)

域 \ 对象	F_1	F_2	F_3	F_4	F_5	F_6	Printer 1	Plotter 2	域 D_1	域 D_2	域 D_3
域D_1	R	R, W								S	
域D_2			R	R, W, E	R, W		W				S
域D_3						R, W, E	W	W			

图7-19 具有切换权的访问控制矩阵

(1) 如图：域d1可以切换到d2，d2可以到d3，但不能反向切

三. 访问矩阵的修改

1. 拷贝权(Copy Right)：打星号的权可以扩展到同列其他域中

1)

域 \ 对象	F_1	F_2	F_3
D_1	E		W*
D_2	E	R*	E
D_3	E		

(a)

域 \ 对象	F_1	F_2	F_3
D_1	E		W*
D_2	E	R*	E
D_3	E	R	W

(b)

图7-20 具有拷贝权的访问控制矩阵

   2) 如图，d1和d2把W*和R*扩展给了d3，但扩展后没了*
   3) 限制拷贝：拷贝时不降拷贝权*扩展，限制访问权进一步扩展
2. 所有权(Owner Right)：增加或删除O所在列各种权的能力

1)

域 \ 对象	F_1	F_2	F_3
D_1	O, E		W
D_2		R^*, O	R^*, O, W
D_3	E		

(a)

域 \ 对象	F_1	F_2	F_3
D_1	O, E		
D_2		O, R^*, W^*	R^*, O, W
D_3		W	W

(b)

图7-21 带所有权的访问矩阵

2) 如图，各种增删权限都可以做到，甚至O所在行本身还可以加一个W*

3. 控制权(Control Right)：Control该行各种权限的增删

1)

域 \ 对象	F_1	F_2	F_3	F_4	F_5	F_6	Printer 1	Plotter 2	域 D_1	域 D_2	域 D_3
域D_1	R	R, W									
域D_2			R	R, W, E	R, W		W				Control
域D_3						R, E	W	W			

图7-22 具有控制权的访问矩阵

2) 如图：d2可以修改d3的权限

四. 访问矩阵的实现

1. 访问控制表(Access Control List)：为每列对象建成一条ACL
   1) 通过不记录空项，减少占用的存储空间，提供查找速度
   2) 一般是每个文件一条ACL，记录有权限的用户进程
   3) 可用于定义缺省访问权集，如果默认ACL找不到该进程想要的权限，才去对象的ACL中查找
2. 访问权限capabilities表：为每行的域都建一张表，每行为对对应对象的权限

1)

	类 型	权力	对 象
0	文件	R--	指向文件 3 的指针
1	文件	RWE	指向文件 4 的指针
2	文件	RW-	指向文件 5 的指针
3	打印机	-W-	指向打印机 1 的指针

图7-23 访问权限表

2) 仅当权限表安全时，保护的对象才可能安全，因此该表只能被通过合法性检查的程序访问

3. 访问控制表和权限表一般是同时存在的，先检查控制表，无权访问便拒绝，并构成异常，有权则建立一权限，连接到该进程，之后，进程便可用该权限访问对象

i.

ii.

iii.

iv.

v. ---------我是底线-------

8外存组织、文件管理

2019年1月2日 14:17

-
- 外存的组织方式

1. 磁盘管理的主要任务是：有效利用存储空间、提高IO速度、提高可靠性

一. 连续组织/分配方式(Continuous Allocation)

1. 通常是一条磁道上连续分配多个盘块，便可不移动磁头地读写
2. 称这些物理文件为顺序文件
3. 保证了逻辑文件中的记录顺序与存储器中盘块顺序一致
4. 目录项的文件物理地址字段中记录首盘块号和长度
5. 外存碎片空间紧凑比内存紧凑慢得多
6. 优点：
   1) 易顺序访问，定长记录文件甚至能随机存取
   2) 顺序访问快，磁头移动距离少，文件访问速度高
7. 缺点：
   1) 碎片空间多，外存空间利用率低，定期紧凑又慢
   2) 必须估计文件长度，估计小不能拷贝，估计大造成浪费
   3) 删插不灵活，要物理移动相邻记录
   4) 难为动态增长的文件分配空间，易导致空间长期空闲

二. 链接组织/分配方式(Chained Allocation)

1) 将文件装到多个离散盘块中，链接成链表，称这种物理文件为链接文件
   (1) 消除了磁盘外部碎片，提高了外存利用率
   (2) 易插删改
   (3) 不用估计文件大小，能适应动态增长

1. 隐式链接：每个目录项中都有指向首盘块和末盘块的指针
   1) 只能依次按下一盘块的指针顺序访问，随机访问效率低
   2) 为提高检索速度和减小指针所占空间，将几个盘块组成一个簇(cluster)
   3) 但以簇为单位又容易增大内存碎片
2. 显式链接：在磁盘中设一张表，显式存放各物理块的指针
   1) 表的序号为物理盘块号，从0开始
   2) 文件的首块号需作为文件地址填入FCB物理地址字段中
   3) 查找中记录的过程在内存中进行，大大减少了访问磁盘次数
   4) 该表即为文件分配表 FAT(File Allocation Table)

(1)

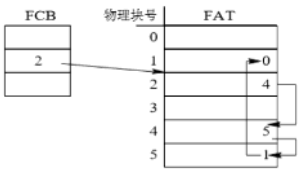


图 6-9 显式链接结构

三. FAT(File Allocation Table)组织方式

1) 卷/分区：将一物理磁盘分成多个逻辑磁盘，每个逻辑磁盘是一个卷
   (1) 卷是被单独格式化和使用的逻辑单元，供文件系统分配空间时使用

(2) 卷中包含文件系统信息、一组文件、空闲空间

(3) 每个卷专门划出一个单独区域存放目录和FAT表和逻辑驱动器字母

(4) 现代OS每物理盘可划分为多个卷，每卷由多个物理磁盘组成

1. FAT12

(1) 由于FAT是文件系统最重要的数据结构，为安全起见，配两张

(2) 只支持8+3格式短文件名

1) 早期以盘块为基本分配单位的FAT12

(1) 每个表项存下一个盘块号，相当于每个结点的指针

(2) 文件只需将首盘块号放入FCB

(3) FAT12允许4096=2^12个表项，若每扇区512字节，则每卷2M

(4)

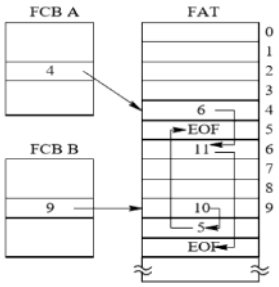


图 6-10 MS-DOS 的文件物理结构

2) 以簇为基本分配单位的FAT12

(1) 在物理上是一组相邻扇区，在逻辑上是FAT一个虚拟扇区

(2) 簇的大小一般是偶数个盘块

2. FAT16：允许65536=2^16个表项，达到减小簇大小同时减小簇内碎片的目的

3. FAT32：允许4294967296=2^32个表项

1) FAT系列最后一个产品，支持长文件名

2) 卷容量不大于8G时，每盘块固定为512b，每个簇为最小8盘块=4k

3) 之后卷最大容量每扩大两倍，簇大小也扩大两倍

4) 只有65537以上个簇的盘才能用FAT32，运行速度慢

5) 不支持4G以上文件的存储，不向下兼容

6)

块大小/KB	FAT12/MB	FAT16/MB	FAT32/TB
0.5	2		
1	4		
2	8	128	
4	16	256	1
8		512	2
16		1024	2
32		2048	2

图 6-11 FAT 中簇的大小与最大分区的对应关系

四. NTFS(New Technology File System)组织方式

1) Windows NT、2000 和 XP 开始才有 NTFS，因此老系统不兼容它

1. NTFS新特征

1) 64位磁盘地址

2) 支持255字符以内长文件名、32767内路径名

3) 有容错功能，故障或差错后仍能正常运行

4) 能保证数据一致性

5) 还有加密、压缩等功能

2. 磁盘组织

1) 以簇为空间分配/回收的基本单位，每簇只属于一个文件，文件可占多簇
2) 卷因子：簇的大小，在磁盘格式化时确定，是整数个扇区
3) 只需管理簇，因而NTFS具有与磁盘块大小无关的独立性
4) 逻辑簇号 LCN(Logical Cluster Number)：以卷为单位，顺序编号卷中所有簇
5) 虚拟簇号 VCN(Virtual Cluster Number)：以文件为单位，顺序编号文件的簇
6) 用文件首簇地址可将VCN映射到LCN
7) 通过计算卷因子*LCN可得偏移量，得物理地址

3. 文件的组织
   1) 以卷为单位，将卷中的所有文件信息、目录信息以及可用空间信息，都以文件记录的方式记录在一张主控文件表 MFT(Master File Table)中
   2) 每个文件作为一条记录，在 MFT 表中占有一行，每行大小固定为 1 KB
   3) 每行为对应文件的元数据(metadata)/文件控制字
   4) MFT表本身也占一行
   5) 数据量小时，元数据可以记录全部数据，减少磁盘访问次数
   6) 数据量大的文件按文件属性排成队列，元数据中存放队列指针

五. 索引组织方式

1) 之前提到的方式的问题：存取慢、FAT需整个调入内存才能找盘块号

1. 单级索引组织方式
   1) 每个文件配一个索引块作为索引表，该文件的所有盘块号计入该块中
   2) 只需在文件目录项中记录该索引块的指针即可直接访问
   3) 大文件不易产生外部碎片了，但小文件的索引块利用率极低
2. 多级索引组织方式
   1) 盘块号装满一块索引块时，需要再申请新索引块，用链指针连接各索引块
   2) 若索引块很多，链接效率会很低，可以用二级索引，记录所有一级索引块
   3) 多级索引加快了查找大型文件的速度，但启动磁盘的次数也增加了
   4) 由于多数系统是中小文件占多数，多级索引效果并不联想

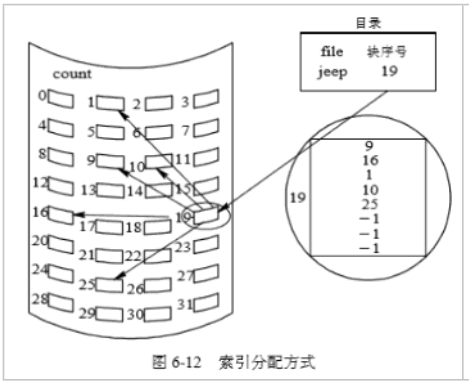


图 6-12 索引分配方式

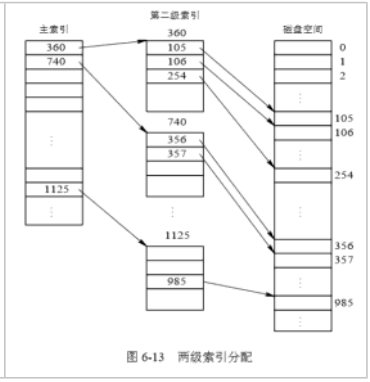


图 6-13 两级索引分配

3. 增量式索引组织方式/混合组织方式
   1) 增量式索引组织方式的基本思想
      (1) 1~10个盘块的小文件可以把盘块首址计入FCB，直接寻址
      (2) 11盘块~几m的中文件可以把一级索引计入FCB，一次间址
      (3) 超大、特大型的文件可以把n级索引计入FCB，n次间址
      (4) 二级索引文件最大长度可达4G，三级可达4T
   2) UNIX System V的组织方式：索引结点中设13个地址项i.addr

(1) 直接地址：i.addr(0)~i.addr(9)存直接盘块号direct blocks

(2) 一次间接地址single indirect：i.addr(10)存索引块

(3) 多次间接地址double indirect：i.addr(11)和12存二级、三级索引

(4)

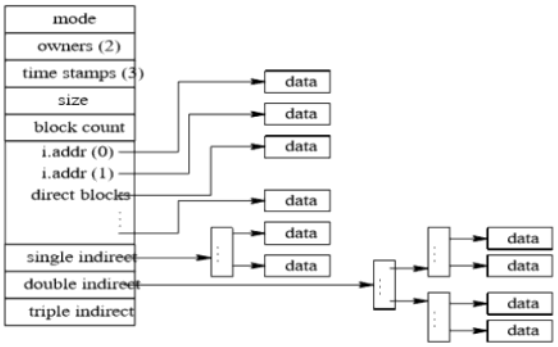


图 6-14 混合索引方式

-
- 文件存储空间的管理

1. 外存可分配存储空间也应设置相应数据结构：磁盘分配表disk allocation table
2. 分配单位应是磁盘块而非字节

一. 空闲表法和空闲链表法

1. 空闲表法
   1) 空闲表：连续分配，每区占一行

(1)

序　　号	第一空闲盘块号	空闲盘块数
1	2	4
2	9	3
3	15	5
4	—	—

图 6-21 空闲盘块表

   2) 存储空间的分配与回收：同内存，首次适应、最佳适应等
   3) 连续分配速度快、访问磁盘次数少；磁头寻道时间少
      (1) 适用于对换空间、小文件；多媒体文件
2. 空闲链表法
   1) 空闲盘块链：以盘块为单位，每盘块都有后继盘块的指针
      (1) 分配时从链首依次摘下合适数量的块；回收时依次挂在末尾
      (2) 过程简单但效率低；盘块为单位可能队伍过长
   2) 空闲盘区链：以盘区为单位，除指针外还应表明当前盘块大小（盘块数）
      (1) 首次适应算法可以用显式链接
      (2) 过程复杂但效率高；队伍段

二. 位示图法

1. 位示图：用1/0对应已/未分配，建立mxn的表，记录mxn个盘块
   1) 占用空间很小，可以全留在内存，常用于微、小型机
2. 盘块分配：顺序扫描到一组连续为0的项，首项在 i 行 j 列（从1开始数）
   1) 首盘块号b = n(i - 1) + j（n为列数）
   2) 再将这些0置1
3. 盘块回收：转换出行号列号，将1置0
   1) i = (b - 1) / n + 1
   2) j = (b - 1) % n + 1

三. UNIX成组链接法

1. 空闲盘块的组织

1) 空闲盘块号栈：存放当前可用盘块号，容量100

(1) 每个栈需要记录空闲盘块数N，N可视作栈顶指针

(2) 栈是临界资源，需设置锁，互斥访问

2) 将空闲盘块分成若干组

3) 每组的N和所有盘块号计入前一组的首盘块的S.free(0)~S.free(99)

(1) 于是首盘块可用于将空闲盘块链成链

4) 第一组的盘块总数和盘块号计入空闲盘块号栈，作为当前可分配盘块号

5) 最后一组的0号盘块存放结尾标志，不视作空闲盘块

(1) 前一组的S.free(0)为0

6)

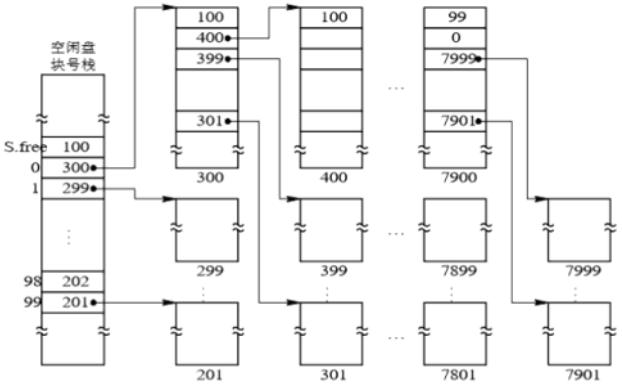


图 6-23 空闲盘块的成组链接法

2. 空闲盘块的分配与回收

1) 分配过程：确认未上锁，取出栈顶空闲盘块号，分配对应盘块，--N，若移到低部S.free(0)则从中取出下一组盘号，调用磁盘读过程将下一组读入栈，此时原栈底盘块的数据不需要了，也可以分配出去了。新进的一组盘块需要分配一组缓冲区

2) 回收过程：将回收的盘块号计入空盘栈顶，++N，N到100时将当前100个盘块号计入下次新回收的盘块中，将其作为新栈底

i.

ii.

iii.

iv.

v.

vi.

vii.

viii.

ix. ---------我是底线-------

8提速、可靠、一致

2019年1月5日 22:00

-
- 提高磁盘IO速度的途径

1. 提高文件访问速度的途径：除了改进目录和选好存储结构外就是提快磁盘IO

一. 磁盘高速缓冲(Disk Cache)

1) 磁盘高速缓存：物理上内存中的缓存区，逻辑上是某些盘块的副本

1. 数据交付(Data Delivery)方式
   1) 数据交互：将高速缓冲的数据传送给请求者进程的内存工作区
   2) 指针交互：指向高速缓存的指针交付给请求进程。节省时间
2. 置换算法：类似请求调页/段：LRU，NRU，LFU，具体需要考虑：
   1) 访问频率：高速缓存的访问频率率=磁盘IO频率<快表频率=指令频率
   2) 可预见性：目录块很少再次访问，未满盘口很可能再次访问
      (1) 可预见会再次使用的应放在LRU链末
   3) 数据一致性：修改数据未拷回磁盘会导致不一致
      (1) 应将需要一致性的数据放在LRU链首，优先写回磁盘
3. 周期性写回磁盘
   1) UNIX专设了一个update程序，定期30s左右调用SYNC，强制将高速缓冲中已修改的数据写回磁盘，防止LRU链末损失30s以上工作量

二. 提高磁盘I/O速度的其它方法

1. 提前读(Read-ahead)：顺序访问文件时，读入下一盘块
2. 延迟写：共享资源挂在空闲区链末，到其他进程申请末区时才写回磁盘
3. 优化物理块分布：尽量将文件安排得不分散，减少磁头移动距离
   1) 如采用位示图、以簇为单位都不会分散；链表就容易分散
4. 虚拟盘RAM：用内存空间仿真磁盘，存放obj等临时文件
   1) 虚拟盘的内容由用户控制；高速缓存的内容系统控制

三. 廉价磁盘冗余阵列(RAID，Redundant Array of Inexpensive Disk)

1) 用一台磁盘阵列控制器管理多个相同磁盘驱动器
2) 大幅增加容量、提快IO、增加可靠性

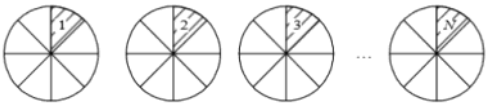
1. 并行交叉存取interleave
   1) 将盘块的数据分成若干子盘块数据，再存储到不同磁盘的同一位置
   2) 并行传输N个磁盘，速度提高N-1倍
   3) 

图 5-29 磁盘并行交叉存取方式

2. RAID 分级
   1) RAID 0。仅提供并行交叉存取。有效地提高磁盘 I/O 速度，但可靠性不好。只要阵列中有一个磁盘损坏，便会造成不可弥补的数据丢失
   2) RAID 1级。有磁盘镜像功能，访问磁盘时，可利用并行读、写特性，将数据分块同时写入主盘和镜像盘。故其比传统的镜像盘速度快，但其磁盘容量的利用率只有 50%，以牺牲磁盘容量为代价
   3) RAID 3级。这是具有并行传输功能的磁盘阵列。它利用一台奇偶校验盘来完成数据校验功能，磁盘的

利用率为 n-1/n。常用于科学计算和图像处理

4) RAID 5级。具有独立传送功能的磁盘阵列。每个驱动器都有独立的数据通路，独立地进行读/写，无专门的校验盘。用来进行纠错的校验信息以螺旋(Spiral)方式散布在所有数据盘上。常用于I/O较频繁的事务处理中
5) RAID 6级。阵列中，设置了一个专用的、可快速访问的异步校验盘。该盘具有独立的数据访问通路，性能改进得很有限，价格昂贵
6) RAID 7 级是对 RAID 6 级的改进，在该阵列中的所有磁盘，都具有较高的传输速率和优异的性能，是目前最高档次的磁盘阵列，但其价格也较高

3. RAID 的优点
   1) 可靠性高。除了 RAID 0 级外都采用了容错技术，可靠性高出了一个数量级
   2) 磁盘 I/O 速度高。并行交叉存取方式，课提高磁盘数目N-1倍
   3) 性价比高。同体积同容量同速度时，牺牲1/N的容量，3倍速度和3倍便宜

◆

◆ 提高磁盘可靠性的技术

1. 影响数据安全性的因素：人为因素、系统因素、自然因素（详见文件保护）
2. 容错技术fault-tolerant tech：通过在系统中设置冗余部件，提高系统可靠性
3. 磁盘容错技术/系统容错技术SFT：增加冗余的磁盘驱动器、磁盘控制器等

一. 第一级容错技术SFT-I
1. 双份目录和双份文件分配表：在不同磁盘备份FAT
2. 磁盘表面少量缺陷后的补救：
   1) 热修复重定向：取约2%磁盘做热修复重定向区，存放发现缺陷时的数据
   2) 写后读校验：写完一块数据马上读出，送去另一缓冲区，比较原数据，若不同，则重写，再不同，则默认该盘块有缺陷，送去热修复重定向区

二. 第二级容错技术SFT-II
1. 磁盘镜像(Disk Mirroring)：在同一磁盘控制器下增设一完全相同的磁盘驱动器
   1) 每次写数据都要写两遍，磁盘利用率低至50%
2. 磁盘双工(Disk Duplexing)：将两磁盘驱动器再分别接到两个磁盘控制器上
   1) 即两个磁盘都有独立通道，可并行读写

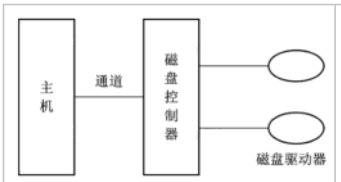


图8-13 磁盘镜像示意图

图8-14 磁盘双工示意图

三. 基于集群技术的容错功能（第三级）
   1) 集群：一组互连的自主计算机组成的计算机系统，人的感觉是一台机器
   2) 即能提高系统并行处理能力，又能提高系统可用性，被广泛使用

1. 双机热备份模式：备份服务器时刻监视着主服务器运行，一旦主服出现故障，备服立刻接替其工作，成为新主服，修复后的原主服会成为新备服
   1) 为连接两台服务器，需要各装一块网卡，建一条镜像服务器链路mirrored server link。如FDDI单模纤允许两服务器距离20公里
   2) 为保证数据同步，需时刻检测主服数据改变，及时用通信系统同步修改到备服的对应数据，为保证通信的高速和安全，一般选高速通信信道
   3) 为保证及时切换，需要配置切换硬件的开关，再备服事先建立好通信配置，迅速处理重新让客户机登录等事宜
   4) 提高了系统可用性，易实现，完全独立，可远程热备份；但备服总是被动等待，系统使用效率仅50%
2. 双机互为备份模式：均为在线服务器，完成不同的任务
   1) 必须先用某专线连接起来，最好再用路由器做备份通信线路
   2) 每台服务器需要两份硬盘，一份装程序，一份接另一台发来的备份数据，即作为另一台的镜像盘
   3) 镜像盘平常对本地用户锁死，保证其数据正确性
   4) 如果通过专线链接检查到了某服务器发生故障，在由路由器验证了故障属实，正常服务器应向连接在故障服务器的客户机广播：需要切换
   5) 而连在非故障服务器上的客户机只会觉得之后网络服务稍慢了些
   6) 故障修复并重新连上网后，服务功能才会被迁回

7) 可扩大到更多台服务器，系统效率很高

图8-15 双机热备份模式

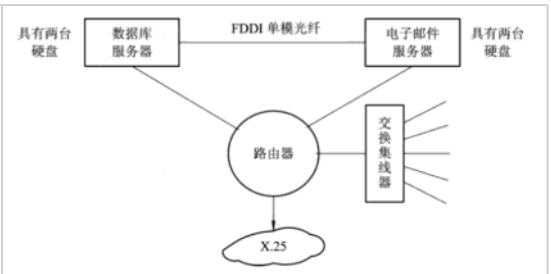


图8-16 双机互为备份系统的示意图

3. 公用磁盘模式：多台计算机连接同一磁盘系统的不同卷
   1) 某计算机故障后，系统调度另一计算机接替，转让卷的所有权
   2) 消除了复制信息的时间，从而减少网络和服务器的开销

四. 后备系统

1) 为防止系统故障和病毒，备份暂时不需要但仍有用的数据和很重要的数据

1. 磁带机：容量大，数十G，便宜；但只能顺序存取，秒速几m

1. 硬盘
   1) 移动磁盘：适用于小型系统、个人电脑。快，脱机方便，保存时间比磁带机稍长，但单位容量贵
   2) 固定硬盘驱动器：适用于大、中型系统。类似双机互备份，每晚将两磁盘的数据互相拷贝到对方的备份区。快，有容错

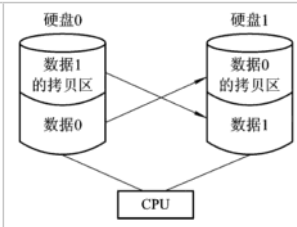


图8-17 利用大容量硬盘兼做后备系统

2. 光盘驱动器
   1) 只读光盘驱动器CD-ROM/DVD-ROM：音视频，不能写，不能后备
   2) 可读写光盘驱动器/刻录机：存数字信息，可读写
      (1) CD-RW刻录机：读写CD、VCD
      (2) COMBO刻录机：读DVD，写CD、VCD
      (3) DVD刻录机：读写CD、VCD、DVD

◆

◆ 数据一致性控制

1. 数据一致性问题：存在不同文件的同一段数据，在任何情况都应相同

一. 事务

1. 事务的定义：访问和修改数据项的程序单位，可视作一系列读写操作
   1) 对不同文件的同一数据读写都结束后才能进行托付操作commit operation/提交操作，结束事务
   2) 任一逻辑错、系统故障导致读写失败后必须进行夭折操作abort operation/回滚操作/取消操作
   3) 夭折事务的数据恢复后，称该事务被退回rolled back
2. 事务的属性（简记为ACID）
   1) Atomic原子性：要么全改，要么不改
   2) Consistent一致性：完成后，所有数据必须一致
   3) Isolated隔离性：并发事务不能在当前事务操作时访问该数据
   4) Durable持久性：事务完成后对系统的影响是永久的
3. 事务记录(Transaction Record)/运行记录log：存储在稳定存储器的数据结构
   1) 包括：事务名、数据项名、修改前旧值、修改后新值
   2) 事务记录表中每行都描述了事务运行时的重要操作：开始、修改、托付、夭折
4. 恢复算法：利用事务记录表处理故障，不致使故障导致非易失性存储器中信息丢失
   1) undo<Ti>：把事务Ti修改过的所有数据恢复为旧值
   2) redo<Ti>：把事务Ti修改过的所有数据设置为新值
   3) 发生故障后，对有开始和托付记录的Ti执行redo；对有开始无托付的Ti执行undo

二. 检查点(Check Points)

1. 检查点的作用
   1) 检查点引入目的：减少发生故障时处理事务记录的视角

2) 隔一段时间将内存中的事务记录保存到稳定存储器、修改过的数据输出到稳定存储器、检查点输出到稳定存储器、执行恢复算法
3) 检查点前托付的Ti不用redo

2. 新的恢复例程算法：查找最近检查点前最后事务Ti，从它开始逐个恢复

三. 并发控制(Concurrent Control)

1) 事务顺序性：各事务修改数据项需按顺序互斥访问
2) 并发控制：实现事务顺序性的技术

1. 利用互斥锁exclusive lock实现顺序性
   1) 每个共享对象设一把互斥锁，访问对象前得先获得其锁，完成后再释放锁
   2) 没获得锁的事务需要等待锁住它的事务释放锁，只能宣布运行失败
2. 利用互斥锁和共享锁shared lock实现顺序性
   1) 若事务只想读对象，发现互斥锁和共享锁都还在，则可获取其共享锁
   2) 若想读，但互斥锁不在，也只能等；若想写，但任一锁不在，也必须等

四. 重复数据的一致性问题

1. 重复文件的一致性
   1) UNIX文件目录项中有ASCII文件名和若干索引结点号，结点数决定重复数
   2) 文件被修改时，必须把其他几个拷贝一起修改
      (1) 查找目录，按索引结点号找到索引，修改该物理位置的数据
      (2) 或建立新的文件拷贝，取代原来的文件拷贝
2. 链接数一致性检查
   1) 共享文件的同一索引结点号可能在目录中多次出现
   2) 需要在索引结点中记录其共享次数count，为0时删除该文件
   3) 可遍历目录，在计数器表中记录索引结点实际出现次数
   4) 若count>实际计数值，则可能永远不删除该文件，浪费空间
   5) 若count<实际计数值，则可能提前删除该文件，造成空索引

i.
ii.
iii.
iv.
v.
vi.
vii.
viii.
ix.
x.
xi.
xii.
xiii.
xiv. ---------我是底线-------

9接口、命令

2019年2月1日 16:52

-
- 用户接口

一. 字符显示式联机用户接口/联机命令接口

1) 指在终端键入命令，终端处理程序接收，终端屏幕反馈
2) 命令：由动词和参数组成，有规定词法、语法、语义和表达形式
3) 命令语言：命令为基本单位，指示操作系统完成特定功能
4) 命令集：诸多命令的集合，完整的命令集包含操作系统提供的全部功能

1. 命令行方式：以行为单位，一般每行不超过256字符，回车符为结束标记
   1) 间断式的串行执行方式：后一个命令的输入需等待前一命令执行结束
   2) 并行执行方式：在命令结尾输入特定标记，将其作为后台命令处理
   3) 简单命令的形式：Command arg1 arg2……
      (1) Command是命令名/命令动词，其余的为执行参数，有时可缺省
2. 批命令方式：将一系列命令组织在文件中，一次建立，多次执行
   1) 如MS-DOS后缀为.bat的文件就是批命令文件
   2) 节省了时间，减少了出错率，方便了用户
   3) 一般操作系统会提供子命令和形式参数书写批命令文件
   4) 如UNIX和Linux的Shell不仅是交互型命令解释程序，还是一种命令级程序设计语言解释系统，允许用户使用Shell简单命令、位置参数和控制流语句编制带形参的批命令文件，称为Shell文件或Shell过程，Shell可自动解释和执行该文件/过程中的命令

二. 图形化联机用户接口Graphics User Interface

1. 其引入：1981年Xerox首次在Star8010推出，1983年苹果也在Lisa和Macintosh上成功使用，之后微软的windows、ibm的os/2、UNIX和Linux的X-Window也使用了GUI。现已有国际GUI标准
2. WIMP技术：Window窗、Icon图标、Menu菜单、Pointing device鼠标
3. Windows 的GUI：系统初始化后，OS为终端用户生产了一个运行explorer.exe的进程，运行一个具有窗口界面的命令解释程序，即桌面，之后点击某程序对应的图标，即会弹出新进程对应的窗口
   1) 是事件驱动控制方式，用户的动作产生事件，驱动程序工作
   2) 由中断系统引出事件驱动控制程序，接收、分析、处理、清除事件
4. 联机命令接口可对资源进行更多更深入的控制，仍受高级用户和程序员的欢迎

三. 联机命令的类型

1. 系统访问类
   1) Login键入管理员处获得的注册名
   2) Password键入密码，一般此时系统会关闭回显，且多次输错会解除连接
2. 文件操作
   1) 显示文件：type、拷贝文件：copy、比较文件：comp
   2) 重命名：Rename

3) 删除：erase。例：参数为*.BAK时删除指定目录所有扩展名为BAK的文件

3. 目录操作
   1) 建立子目录：mkdir
   2) 显示目录项：dir
   3) 删除空子目录：rmdir（只包含.和..）
   4) 显示目录结构：tree
   5) 改变当前目录：chdir。例：参数为..时会返回到上级目录
4. 其他
   1) 输出重定向：>文件或设备、输入重定向：文件或设备<
   2) 管道连接：命令|命令
   3) 过滤。如MS-DOS的find/N输出含有指定字串的行，find/C输出其行数，find/V输出不含指定字串的行数
   4) 批命令。如MS-DOS中用batch命令执行批命令文件

- ◆
- ◆ Shell命令语言

1. Linux的Shell是命令语言、命令解释器程序、程序设计语言的统称
   1) 作为命令语言，有自己内建的命令集，向用户提供操作系统的接口
   2) 作为设计语言，支持函数、变量、数组、程序控制结构
   3) 作为命令解释器程序，可对输入的命令解释执行

一. 简单命令简介
   1) 简单命令：一条命令行仅有一个命令
   2) 一条简单命令便是一个目标程序的名字

1. 简单命令的格式
   1) UNIX和Linux都规定用小写字母
      (1) 选项：-开始的，后跟多个字母或数字的可选自变量
      (2) $：Linux的默认系统提示符
      (3) 例：$ ls -tr file1 file2按最近修改序和反字母序打印两目录的目录项
2. 简单命令的分类
   1) 可分为系统提供的标准命令：调用各种语言处理、实用程序等，管理员可增添，和用户自定义的命令
   2) 也可按是否常驻内存分为内部命令、外部命令。如改变工作目录cd是内部的，拷贝cp，移动rm是外存某目录上的
   3) 简单命令的管理是对用户透明的，搜索路径能找到对应应用程序的即可视作系统调用，由Linux内核处理
3. Shell的种类
   1) Bourne Shell，简称B Shell。是UNIX最初使用的Shell。比C Shell小，效率高，交互性略差
   2) Bourne Again Shell简称Bash。是B Shell的一个版本，扩展了命令补全、命令编辑、命令历史表等功能。灵活强大友好，是Linux默认的
   3) C Shell，简称C Sh。更适合编程，是标准Berkeley System Distribution

命令解释。语法类似C。提示符为%。兼容B Shell，提供更多特殊功能，如!表示重复执行，!!表示重复执行最后输入的命令

4) Tcsh，是C Shell的一个扩展版本，包括命令行编辑、可编程单词补全、拼写矫正、历史命令替换、作业控制等，是Linux的默认C Shell
5) Korn Shell，简称K Sh，结合了B Shell和C Sh的有点，和B Shell兼容
6) Pdksh，是Linux上的K Sh扩展，支持在命令行上挂起、后台执行、唤醒、终止程序

二. 简单命令的类型

1. 进入/退出系统Login/Logout
   1) 进入系统/注册，管理员用用户名，在系统文件树建立子目录树根结点
   2) 退出系统，由系统为用户记账，清除用户使用环境，一般是按ctrl d
2. 文件操作：显示内容cat、复制cp、改名mv、撤销rm、确定类型file
3. 目录操作：建立子目录mkdir、撤销空目录rmdir、改变工作目录cd
4. 系统询问：
   1) 日期时间date（可带参数，用于修改）
   2) 当前所有用户名、终端名、注册时间who（选项-L显示当前用户数）
   3) 显示当前目录路径名pwd
5. 重定向、管道、通信、后台等

三. 重定向与管道命令

1. 重定向命令
   1) Login程序自动设置标准输入为文件为键盘，标准输出文件为屏幕
   2) $ cat file1>file2将显示文件file1改为复制到file2
   3) $ cat file4>>file2将file4数据添加到file2现有数据后
   4) $ wc<file3将统计键入字数改为统计file3字符数
   5) a.out<file1>file0表示可执行文件a.out重定向为从file1提取数据，结果输出到file0
2. 管道命令：建立通道pipe文件，缓冲前一命令的输出，作为后一命令的输入
   1) 单向性：只从前一命令输入，给后一命令读取
   2) 同步性：管道满时，前一命令暂停，管道空时，后一命令暂停
   3) 例：$ cat file|wc将file的数据给wc命令计数

四. 通信命令

1. 信箱通信命令mail：非交互非实时式通信
   1) 以注册名命名私有信箱，以接受者注册名命名信件，存在/usr/spool/mail
   2) mail命令后的参数为接收者注册名，新行键入信件正文
   3) .或^D结束输入，读信选项为r：先进先出读信、q：按中断字符（del或return）退出信箱且不改变信箱内容、p：显示全部信件等（不用p选项的话每显示完一个信件会询问一次是否读下一条）
2. 对话通信命令write：实时与当前系统其它用户联机通信
   1) UNIX上允许一个用户在多终端上注册，所以要先用who命令确认
   2) 再用write user[ttyname]（只注册了一个终端时可缺省终端名）

3) 之后对应终端会提示原用户名和终端名

3. 允许或拒绝接受消息的mesg命令
   1) n选项表示拒绝写，如正在联机编写重要资料时一般选n
   2) y选项表示恢复写许可
   3) 不带选项表示只报告当前状态而不改变

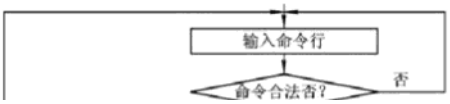
五. 后台命令
1. 后跟&的命令会被Shell后台执行，其标准输入文件会被定向到/dev/null空文件
2. 可用ps、wait、Kill命令了解和控制后台进程的运行

-
- 联机命令接口的实现

一. 键盘终端处理程序
1. 字符接收功能
   1) 面向字符：直接将ascii码传给以后程序
   2) 面向行：收到行结束符后将行缓冲送给命令解释程序（有时存的是键码，如按下a键后，其键码30存再IO寄存器，由终端处理程序转换成ascii码）
2. 字符缓冲功能
   1) 专用缓冲：每个终端各200字符左右的缓冲区，适用于终端较少的情况
   2) 公用缓冲：每个缓冲区20字符左右，链成链，数据传给程序后重新链入空缓冲区链，提高缓冲利用率
3. 回送显示：硬件实现较快但不灵活；软件实现方便大小写切换和密码输入
4. 屏幕编辑：删除字符（移出字符队列末字符、如backspace或ctrl h）、删除行、插入、移动光标、屏幕上卷或下移等
5. 特殊字符处理
   1) 中断字符：如break、delete、ctrl c，中断处理程序将发出软中断信号，各进程收到后自我终止
   2) 停止上卷、恢复上卷：如ctrl s、ctrl q并不会被存储，而是用于设置中断数据结构的上卷标志，终端视图输出新数据前需检查该标志

二. MS-DOS解释程序
   1) 命令解释程序一般放在用户层，以用户态运行
   2) MS-DOS是微软1981开发的微机OS，其命令解释程序为COMMAND.COM

1. 命令解释程序的作用：提示用户键入命令，并读入、识别、处理命令，转交控制权给对应处理程序
2. 命令解释程序的组成
   1) 常驻部分：正常退出中断INT20、驻留退出中断INT27、错误退出中断INT24；检查程序终止后是否被覆盖、若被覆盖则重新调入内存
   2) 初始化部分：每次系统接电或重启后找到AUTOEXEC.BAT文件并装入基址执行，因为只需初始化一次，第二个装入的文件就会覆盖它
   3) 暂存部分：命令解释程序等，可被用户程序覆盖，运行完后再从磁盘调回

命令解释程序的工作流程：系统接电后控制权先	

3. 交给初始化部分，AUTOEXEC.BAT执行完后转交控制权给暂存部分，读入键盘缓冲命令，判断出对应内部命令并转交控制权或通过系统调用exec装入该外部命令，再转交

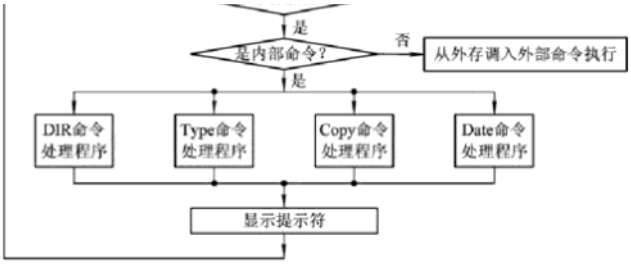


图9-2 COMMAND.COM的工作流程

三. Shell解释程序

1) Shell是Linux或UNIX的系统最外层，所有命令都是经过Shell的解释/识别再传给内核的程序处理的

1. Shell命令的特点
   1) 一条命令行可以有多个命令，解释后可能要产生多个命令处理进程
   2) 命令间可能有不同分隔符，如;顺序执行、&后台执行、|管道执行
2. 二叉树结构的命令行树
   1) 命令表型结点：;和&分隔符作为结点，左右部分构成其左右子树
      (1) ;结点是递归执行完左子树才能执行右子树
      (2) &结点是启动左子树后即可执行右子树
   2) 管道文件型结点：|分隔符作为结点，左右部分构成其左右子树
   3) 简单命令型结点：是内部命令就直接执行，否则建个新子进程，运行完毕后再恢复Shell执行
3. Linux命令解释程序的工作流程
      (1) 内核为每个终端用户建立一个进程执行Shell解释程序
   1) 读取键入的命令行
   2) 分析命令、建立二叉树、以命令名为文件名、将参数改造为系统调用execve内部处理时要求的形式
   3) 终端进程调用fork为每一命令建立子进程
   4) 等待子进程完成。;结点是通过系统调用Wait4()，调用execve()查找对应程序调入内存执行，结束后唤醒父进程执行下一条；&结点启动左子树后可立刻执行右子树

5)

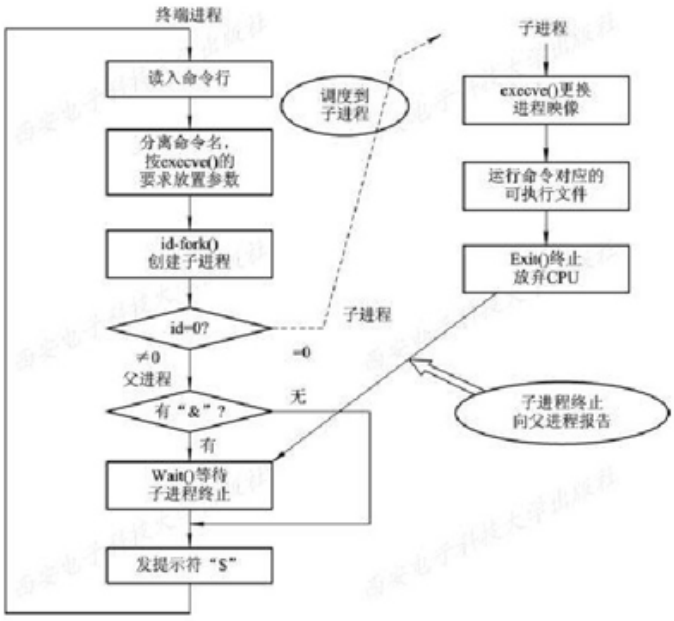


图9-5 Shell基本执行过程及父子进程之间的关系

i.
ii.
iii.
iv.
v.
vi.
vii.
viii. ---------我是底线-------

9系统调用

2019年2月8日 16:30

-
- 系统调用的概念和类型

1. 程序接口是OS给用户程序设置的取得OS服务唯一途径，由系统调用构成
2. 系统调用system call：既提供了用户程序和OS内核的接口，又可给OS自身使用，每个系统调用都是能完成一特定功能的子程序

一. 系统调用的基本概念

1. 系统态/核心态和用户态
   1) 特权指令：系统态运行的指令，只能OS使用。既能访问用户空间，也能访问系统空间。启动外部设备，设置时间，关中断，执行状态转换等
   2) 非特权指令：用户态运行的指令，为防止程序异常破坏系统，不可直接访问系统硬件和软件，只能完成一般性的操作和任务
   3) 程序使用特权指令会发出权限错ixnh，系统转入错误处理程序，停止运行该程序，重新调度
2. 系统调用与一般过程调用的区别
   1) 主调程序在用户态，被调程序却在系统态
   2) 需要先通过软中断机制切换到系统态，再经内核分析，才能转向调用
   3) 抢占式调度系统中，调用的过程执行完后可能根据优先级执行其他进程
   4) 嵌套调用一般有深度限制，如有的系统最大深度为6
      (1) 例：拷贝文件，再指定文件名后，需要依次调用
      open、creat、alloc、read、close、write、close，任一调用出错都需要调用exit正常结束程序
3. 中断机制。如MS-DOS的INT21H
   1) 系统调用都是通过中断机制实现的，每个系统调用都通过中断入口实现
   2) 应当是被保护的，如IBM PC上Intel提供了多达255个中断号，未授权给应用程序保护等级的中断号被应用程序调用后会引起保护异常，导致被终止；Linux则只给了3，4，5，80h四种中断号，第四个是系统调用中断号

二. 系统调用的类型

1. 进程控制类：创建和终止进程、获得和设置进程属性、等待某事件等
2. 文件操作类：创建删除、打开关闭、读写等
3. 进程通信类：基于连接的消息传递、基于虚地址空间共享存储区的通信
4. 设备管理类：申请释放、设备IO、重定向、获得设置属性等
5. 信息维护类：获得时间、获得操作系统版本、获得当前用户、获得空间大小等

三. POSIX标准Portable Operating System IX：基于UNIX的可移植操作系统接口

1. 定义了标准API、保证程序源代码可兼容多系统移植运行
2. 定义了一组构造操作系统必须的过程，大多数系统调用应对应一个或一组过程
3. 并没有指定系统调用的实现形式，早期流行汇编，新推出的系统中常用C语言写系统调用，以库函数形式提供，隐藏了访管指令的细节

4.

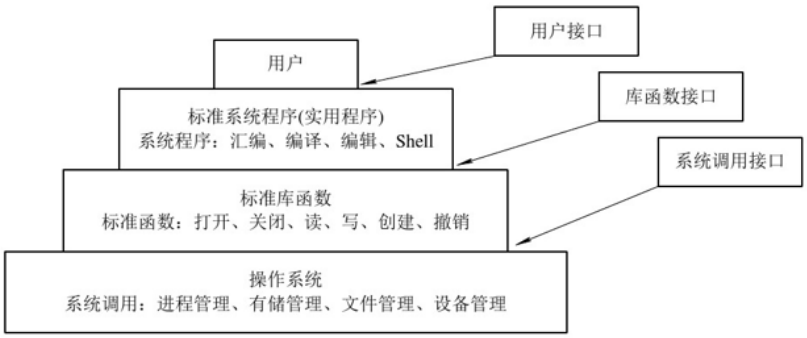


图9-7 UNIX/Linux系统程序、库函数、系统调用的分层关系

-
- UNIX系统调用

一. 进程控制

1. 进程的创建和终止
   1) 创建子进程fork，继承调用者进程的各种环境、文件、根目录、当前目录、大多数属性，进程映像也基本相同
   2) 终止进程exit，创建子进程时一般在末尾安排一exit，使其自我终止，留下一条记账信息status
2. 改变进程映像和等待
   1) 执行文件exec，覆盖调用者的进程映像
   2) 等待子进程结束wait，将调用者挂起，终止与子进程的同步
3. 其他进程调用
   1) 获得进程ID，如getp-id活动标识符、getpgrp活动进程组id，getppid获得父进程id
   2) 获得用户id，如getuid获得真正id，geteuid获得有效id，getgid获得真正用户组id
   3) 进程暂停pause，挂起进程直至收到信号

二. 文件操纵

1. 文件的创建和删除
   1) 创建或重写同名文件creat，并打开，返回其描述符fd，失败则返-1
   2) 删除文件在UNIX无法通过对应系统调用实现，无连接时才能删
2. 文件的打开和关闭
   1) 打开open将文件从硬盘拷到内存，返回描述符fd，作用是代替路径名
   2) 关闭close断开程序与文件的快捷通路，访问计数为0后才能真正关闭
3. 文件的读和写
   1) read和write都需要提供三个参数：描述符fd、缓冲区首址buf（读的目标地址或写的源地址）、需传送字节数nbyte
4. 建立与文件的连接和去连接
   1) 连接link，实现文件共享，并给连接用户进程数+1
   2) 去连接unlink，实现断开连接，并给连接计数-1，为0后才能删除

三. 进程通信和信息保护

1) 软件包IPC专门用于实现进程通信，包括消息机制、共享存储区机制、信号量机制，每个机制中都提供了对应的系统调用

1. 进程通信
    1) 消息机制：msgget建立信息队列，成功则返回消息队列描述符msgid用于访问该消息队列；msgsend用于发生消息给队列；msgrcv用于从指定队列接收指定类型的消息
    2) 共享存储区机制：先用shmget建立共享存储区，成功则返回共享存储区描述符sgmid；shmat将该共享区连接到进程的虚地址空间上；shmdt拆除进程与共享存储区间的连接
    3) 信号量机制，同二章，可将一组信号量形成一个信号量集，执行原子操作
2. 信息维护
    1) 设置时间stime，获得时间time
    2) 获得进程和子进程使用CPU时间times，包括在用户空间执行指令时间、调用进程时间、子进程在用户空间使用CPU时间、系统为各子进程花费CPU时间，这些时间可填写到指定缓冲区
    3) 设置文件访问和修改时间utime，如果参数times为NULL，有写权限的以后可修改为当前时间，否则times为执行utim buf结构的指针，以后只能讲访问时间和修改时间置入该结构中
    4) 获得UNIX系统名称uname，相关信息存在utsname结构中，包括系统名字符串、系统在网络中的名称、硬件的标准名称等

-
- 系统调用的实现

一. 系统调用的实现方法

1. 系统调用号和参数的设置
    1) 一般每条系统调用对应唯一系统调用号
    2) 在系统调用命令（陷入指令）传递调用号给中断和陷入机制的方法
        (1) 直接放在系统调用命令（陷入指令）中，如IBM370和早期UNIX
        (2) 装在寄存器中，如MS-DOS放AH，Linux放EAX
    3) 将系统调用所需参数传给陷入处理机构和系统内子程序/过程的方法
        (1) 让陷入指令自带少量有限参数
        (2) 用相应寄存器，传递有限数量的参数，如MS-DOS用MOV指令
        (3) 参数表方式，只将参数表指针存进寄存器，如UNIX和Linux
2. 系统调用的处理步骤
    1) 设置完调用号和参数后，UNIX会执行CHMK命令，MS-DOS会执行INT-21软中断
    2) 首先处理状态由用户态转为系统态，由硬件和内核程序进行系统调用的一般性处理，主要是转移上下文到堆栈和保存参数到指定地址
    3) 之后根据调用号查系统调用入口表，以转入对应子程序
    4) 最后恢复CPU现场，继续往下执行
3. 系统调用处理子程序的处理过程

1) 系统调用的功能主要是由系统调用子程序来完成的
2) 如Creat命令的子程序中，核心会根据文件路径名查找指定目录，如果已存在无写权限的文件，会认为出错；如果已存在有权限的文件，就释放该盘块，准备写入新数据文件；如果无指名文件，则表示要创建一个新文件，核心需要找出一个空目录项，初始化，再打开新建文件

二. UNIX系统调用的实现

1) UNIX系统V的内核中有一个trap.S文件，它是中断和陷入总控程序，用于一般性处理中断和陷入。为了效率，由汇编语言编写
2) 还有一个C语言编写的trap.C程序，专门处理系统调用、进程调度中断、跟踪自陷非法指令、访问违章、算术自陷等12种陷入的公共问题。主要包括确定系统调用号、传送参数、转入相应子程序等

1. CPU环境保护
   1) 用户态，执行CHMK命令前，应填好参数表，将地址传进R0寄存器
   2) 执行CHMK命令后处理机转为核心态，硬件自动将处理机状态字长PSL、程序计数器PC、代码操作数code等压入以后核心栈，再转入trap.S
   3) trap.S执行后，将陷入类型type和用户栈指针usp压入用户核心栈，再通过特定寄存器的屏蔽码，把对应寄存器中的CPU环境也压入栈
2. AP和FP指针
   1) AP指向参数表，FP指向调用栈帧，指示本次系统调用所保存的数据项
   2) 出现新系统调用时，需将AP和FP303压入栈，实现了嵌套系统调用
   3) trap.S完成了CPU环境和AP、FP的保存后，调用trap.C完成后续
3. 确定系统调用号
   1) trap.C的调用形式为trap(usp,type,code,PC,PSL)
   2) 令i=code&0377。0<i<64时i即为系统调用号，i为0时需通过间接指针
4. 参数传送
   1) 指由trap.C将系统调用参数表的数据从用户区传到User结构的U.U-arg中
   2) 根据系统调用定义表规定的参数个数进行传送，最多10个
5. 利用系统调用定义表Sysent转入相应的处理程序
   1) 该表是个结构数组，每个结构里有所需参数数、待传参数数、子程序入口
6. 系统调用返回前的公共处理
   1) UNIX进程动态优先级随时间加长而降低，每次系统调用返回trap.C都需重新计算优先级。另外，系统调用执行时发生错误时会设置再调度标志，处理子程序在计算优先级后若检查到该标志，便会调用switch调度程序，选择就绪队列的最高优先级进程，转交处理机给它运行
   2) 当进程处于系统态时，不理睬其他进程发来的信号；回到用户态时，内核才坚持该进程是否有收到信号并执行相应动作。处理结束后执行返回指令RET，将已被压入用户核心栈的数据退还给相应寄存器、让进程继续执行

三. Linux系统调用

1. 和UNIX相似。最多有190个系统调用
2. Linux在CPU的保护模式下提供了四个特权级别，目前内核都只用到两个：0级

内核态和3级用户态

3. 用户对系统调用不能任意拦截和修改，以保证内核安全
4. 每个系统调用的组成部分
    1) 内核函数，作为核心驻留在内存，由C书写，运行在内核态，一般不能调用其他系统调用或应用程序可用的库函数
    2) 接口函数，是提供给应用程序的API，以库函数形式存在lib.a中，由汇编语言书写，主要功能是把系统调用号、入口参数地址传给相应核心函数，并使用户态下运行的应用程序陷入核心态
5. 由汇编写的系统调用入口程序entry(sys_call_table)
    1) 包含了系统调用入口地址表，给出了系统调用核心函数名
    2) 每个核心函数的编号定义在include/asm/unistd.h：

        ENTRY(sys-call-table)

            long SYMBOL_NAME(sys_xxx)i
    3) Linux的系统调用号即系统调用入口表中位置序号
    4) 所有系统调用通过接口函数将调用号传给内核，内核转入系统调用控制程序，通过调用号定位核心函数，Linux内核陷入由0x80(int80h)中断实现
6. 系统调用控制程序的工作流程：取系统调用号，检验合法性；执行int80h产生中断；转换地址、切换堆栈、转内核态；中断处理，通过调用号定位内核函数；通过寄存器内容从用户栈取入口参数；执行核心函数，结果返回给程序

四. Win32的应用程序接口

1. 应用程序接口API是一个函数的定义，说明如何获得一个给定的服务；而系统调用是通过中断向内核发出的一个请求。API可能调用也可能不调用系统调用
2. Windows程序设计模式是事件驱动方式，与UNIX和Linux有根本的不同，主程序需要等待事件的发生，根据事件的内容，调用相应的程序
3. 通过调用Win32API可以创建文件、进程、线程、管道等对象，并将聚彬返回给调用者，调用者课根据句柄间接知道对象在内存的具体位置
4. Windows中，只有对操作系统性能起关键作用的程序才能运行在核心态，如对象与安全管理器、线程与进程管理器、虚存管理器、高速缓存管理器、文件系统等，它们构成了操作系统执行体executive
5. 在Intel x86处理机上，当程序调用操作系统服务时，需要执行int2E指令，由硬件产生陷入信号，系统捕捉后切换到核心态，控制权转交给陷入处理程序的系统服务调度程序，该程序负责关中断、保存现场、检查参数、并转移到核心态堆栈、查找系统服务调度表/陷入向量表以获得对应服务的地址并转交控制权
6. 支持API的三大组件Kernel、User、GUI

    (1) Windows将这三个组件置于DLL动态链接库dynamic link library中

    (2) 任何应用程序都共享这三个模块的代码，可直接调用其函数

    1) Kernel包含了大量操作系统函数，如内存、进程的管理等
    2) User集中了窗口管理函数，包括创建、撤销、移动、对话等
    3) GUI提供画图、打印等函数

        i.

ii.
iii.
iv.
v.
vi.
vii.
viii. ----------我是底线-------

10多处理机系统

2019年2月13日 14:37

-
- 多处理机系统的基本概念

1. 提高计算机系统性能的主要途径：
   1) 提高元器件的运行速度，尤其是处理器芯片
   2) 改进系统的体系结构，尤其是引入多处理器
2. MPS多处理机系统Multiprocessor System是20世纪70年代出现的

一. 多处理机系统的引入

1. CPU时钟频率问题
   1) 早期提高CPU时钟频率可提供计算速度，随芯片制造工艺水平提供，CPU的时钟频率已从每秒数十次发展到数兆赫兹GHz
   2) 然而这种方法受限于CPU指令和数据以电子信号的方式通过介质送入送出的传输时间。电子信号早真空的速度为30cm/ns，在铜线或光纤中的传输速度约20cm/ns，100GHz计算机的信号路径便不得超过2mm
   3) 而元器件缩小又使散热成为一个棘手的问题，高端Pentium系统的CPU散热器体积已超过了其本身的体积
2. 增加系统吞吐量：增加处理机数目使系统在单位时间内能完成更多工作
3. 节省投资：达到相同处理能力的情况下，多处理机系统更省外设、内存等
4. 提高系统可靠性：将故障处理机的任务迁移到其他处理机的系统重构功能

二. 多处理机系统的类型

1. 紧密耦合MPS和松散耦合MPS
   1) 紧密耦合Tightly Coupled：通过高速总线或高速交叉开关实现多处理机互连，所有资源和进程都由操作系统实施控制管理
      (1) 多处理器共享整个主存和IO设备，访问时间约10~50ns
      (2) 多处理器与多存储器分别相连，或将主存划分成独立访问的模块
         i. 实现了处理机同时访问主存
         ii. 处理机间的访问采用消息通信方式，在10~50μs内发出
         iii. 互连较慢，软件实现较复杂，但使用构件方便
   2) 松散耦合Loosely Coupled：通过通道或通信线路实现互连，每台计算机有各自的存储器和IO设备，配置了OS管理本地资源和本地进程，独立工作，必要时才交换信息协调工作。消息传递一般需要10~50ms
2. 对称多处理器系统SMPS和非对称多处理器系统ASMPS
   1) Symmetric Multiprocessor System：各处理器单元的功能和结构都相同，是最常见的MPS，如IBM的SR/600 Model F50用了4片Power PC
   2) Asymmetric Multiprocessor System：有多重类型的处理单元，功能结构各不相同。系统中只有一个主处理器，有多个从处理器

-
- 多处理机系统的结构

1. 共享存储器的MPS中，若干处理器共享一个RAM，系统需要为每个CPU的程序提供一个完整的虚拟地址空间视图，每个存储器地址单元均可被所有CPU读写
    1) 方便了处理机间的通信
    2) 需在进程同步、资源管理及调度上，做出有别于单处理机系统的特殊处理
    3) 进程对不同存储器模块的读写速度存在的差异，形成了不同的结构：
        (1) Uniform Memory Access：统一/一致性内存访问
        (2) Nonuniform Memory Access：非统一/非一致性内存访问

一. UMA多处理机系统的结构

1) 各CPU的功能结构都相同，无主从之分，对各存储器单元的读写速度相同
2) 是一种SMPS，按处理机与存储器模块的连接方式不同，可分为：

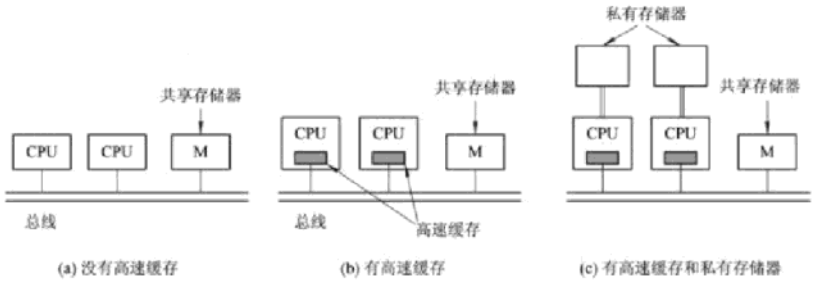
1. 基于单总线的SMP结构/均匀存储器系统
    1) 所有处理器通过公用总线访问同一物理存储器，只需运行操作系统的一个拷贝，单处理器系统的程序可直接移植，只要总线空闲，CPU即可将存储器地址放到总线上并插入若干控制信号，等待存储器将所需内容放上总线
    2) 缺点是可伸缩性有限，总线资源的瓶颈效应限制了CPU数目难以超过20
    3) 可通过在CPU内部、板上、附近等地设置高速缓存，减少其对总线的访问频率，大幅减少总线上的数据流量，支持更多CPU，高速缓存交换和存储单位一般为32或64字节块
    4) 

图10-1 基于总线的SMP结构

2. 使用多层总线的SMP结构
    1) 总线瓶颈问题的进阶解决方法是给每个CPU再配一个本地私有存储器
    2) 各CPU的本地总线负责连接私有存储器、IO设备、系统总线
    3) 系统总线一般在通信主板实现，用于访问共享存储器和连接CPU本地总线
    4) 一般只将共享变量放在共享存储器中，减少系统总线的流量，支持16~32个CPU，但这样提高了编译器的要求，编程时对数据安排的难度也高了
3. 使用单级交叉开关的系统结构
    1) 交叉开关crossbar switch类似电话交换系统，将所有CPU结点和存储器结点通过交叉开关阵列相互连接，每个交叉开关均为两个结点提供一条专用连接通道，避免了链路的争夺，方便了CPU间的通信
    2) 闭合状态称为开；打开状态称为关

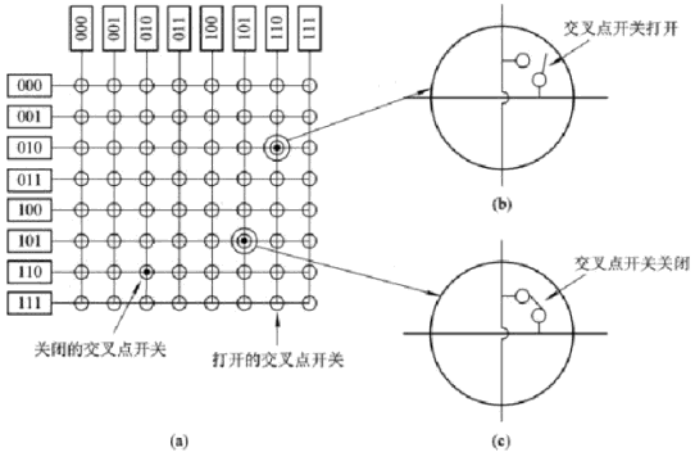
3) 

图10-2 使用交叉开关的UMA多处理机系统

4) 使用交叉开关的UMA多处理机系统的特征

(1) 结点间的连接：一般是NxN的阵列，每列都只能开一个开关

(2) CPU结点与存储器之间的连接：为支持并行存储访问，一行可同时接通多个开关

(3) 交叉开关的成本为N方，端口数N一般不超过16

4. 使用多级交换网络的系统结构

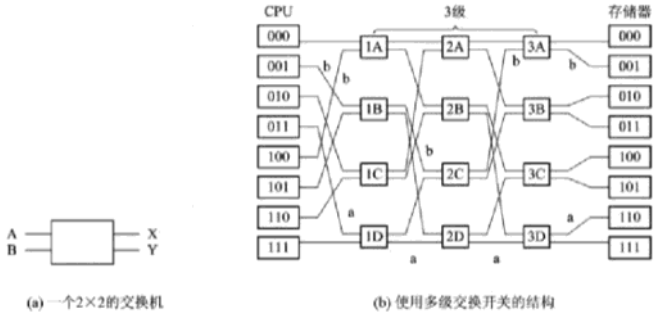
1) 

图10-3 使用多级交换网络的SMP结构示意图

2) 连接多个图a所示交叉开关即可实现多级交叉开关网络，每个最小交叉开关都是一个交叉开关级，处理机和存储器模块分别位于网络两侧

3) 所有处理机访问方式都一样，机会均等，多条路径减少了阻塞概率、分散了流量，提高了访问速度；但硬件结构昂贵，处理机数一般也不超过100

二. NUMA多处理机系统结构

1) 上述UMA的SMP体系都有共享性，随CPU数量增加，路径流量急剧增加，形成超载，使内存访问冲突迅速增加，制约了系统性能，浪费了CPU资源，降低了CPU性能的有效性，为有效扩展，需要利用NUMA技术

1. NUMA结构和特点

1) NUMA系统访问时间随存储字的位置不同而变化，公共存储器和分布在所有处理机的本地存储器共同构成了系统的全局地址空间

2) NUMA拥有多个处理机模块/节点，各节点间通过一条公用总线或互连模块进行连接和信息交互，可包含多达256个CPU

3) 各节点又可由多处理机组成，如四个拥有各自独立本地存储器、IO槽口的奔腾处理机，可通过一条局部总线和一个单独主板的群内共享存储器连接

4) NUMA结构特点：共享存储器在物理上分布式，在逻辑上连续，存储器

的集合就是全局地址空间，每个CPU都可访问所有内存，但指令不同

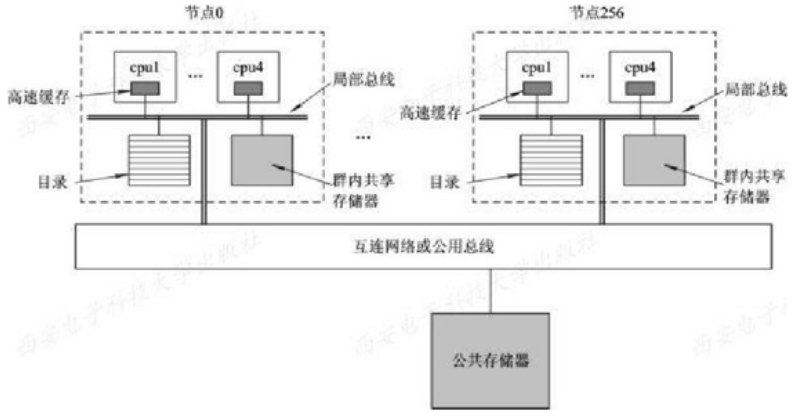
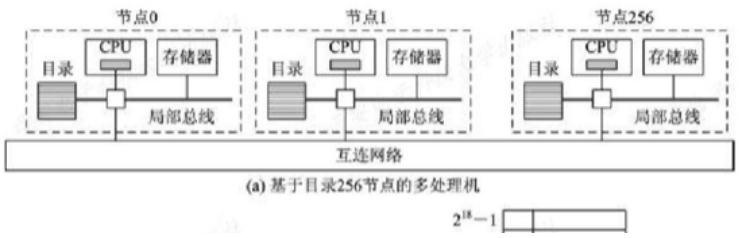
5) NUMA存储器分三层：本地存储器、群内共享存储器、全局共享存储器或其他节点存储器。访问本地存储器最快，访问属于其他处理机的远程存储器较慢。所有机器有同等访问公共全局存储器的权利，不过访问群内存储器快于访问公共存储器
6) 运行在NUMA系统的程序按址寻址数据时，首先察看本地存储器，再是本节点的共享存储器，最后是其他节点的远程内存（全局共享存储器）
7) 为更好地发挥系统性能，应尽量减少不同节点间的信息交互。各CPU都配备了高速缓存的NUMA称为CC-NUMA，没有配的称为NC-NUMA
8) 

图10-4 NUMA结构的多处理机系统

2. CC-NUMA构造方法
   1) 基于目录的多处理机思想：同一CPU拥有的若干高速缓存单元，以一定数量为一组，构成一个高速缓存块，为每个CPU配置一张高速缓存块目录表，记录和维护每个缓存块的位置和状态，读写高速缓存、变换高速缓存块节点、修改目录、访问存储器单元等操作前需要先查表
   2) 一般每个表项记录一个本地高速缓存块地址，每个高速缓存单元的内容是某个本地存储器单元内容的拷贝
   3) 32位机适合16位表项长度，将存储器空间划分为若干长度为2^(22-16)的存储器单元组，对应地目录项也应有2^(32-8-6)个
   4) 当CPU发出远程存储器单元访问指令后，由操作系统的MMU翻译出物理地址，将请求消息通过互联网络发送给对应结点，询问其对应块的地址，该节点收到请求后，将对应块内容送进请求CPU的高速缓存中
   5) 若该内容已进入其他节点的高速缓存，则会通知该节点将该内容送给发出请求的节点，之后这三个节点需要修改本地目录的对应内容指向新位置
   6) 可见，这种NUMA结果需要大量消息传递实现存储器共享，访问远程内存的延时远超本地内存，一个存储块只能进入一个节点的高速缓存限制了存储器访问速度的提高，CPU数量增加无法使系统性能线性增加



(a) 基于目录256节点的多处理机

7)

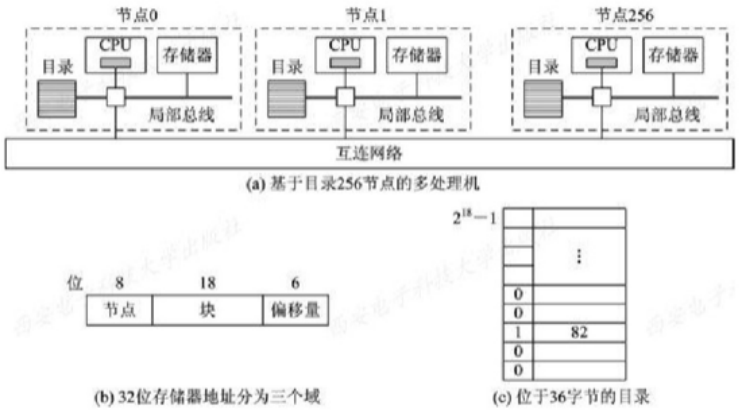


图10-5 CC-NUMA构造方法

-
- 多处理机操作系统的特征与分类

一. 多处理机操作系统的特征

1. 并行性：依赖于系统结构的改进，多处理机可使多个进程并行执行，不过每个实处理机依旧可用多道程序技术虚拟为多个虚处理机，实现并发
2. 分布性：对任何结构的多处理机系统，任务、资源及对它们的控制等方面都呈现出一点的分布性，尤其在松散耦合系统中
    1) 任务：只要能把作业分为多个可并行执行的子任务，便可分给多个处理机
    2) 资源：各处理机都可拥有存储器、IO设备等本地资源
    3) 控制：各处理单元可配置自己的操作系统，控制管理本地及协调通信共享
3. 机间的通信和同步性：不同处理机之间的同步和通信也对提高并行性、改善系统性能至关重要，且实现机制复杂得多
4. 可重构性：某处理机或存储模块等资源发生故障时，系统需要自动切除故障，换上备份资源，对系统重构，保证继续工作

二. 多处理机操作系统的功能

1. 进程管理
    1) 进程同步：需要额外解决不同处理机并行执行时引发的同步问题
    2) 进程通信：处理机间的通信信道较长，甚至要通过网络。一般是间接通信
    3) 进程调度：了解各处理机适合什么任务，各作业间的关系，哪些任务需顺序执行，哪些可并行，各任务需要什么资源，实现负载的平衡
2. 存储器管理：除了地址变换机构和虚拟存储器外，还需
    1) 地址变换机构需要确认物理地址是否远地存储器
    2) 访问冲突仲裁机构：多处理机竞争同一存储器模块时需要按规则决定顺序
    3) 数据一致性机制：确保共享主存的数据在多个本地存储器中拷贝一致
3. 文件管理
    1) 集中式：所有处理机的所有用户文件集中在某处理机的文件系统中
    2) 分散式：各处理机配置各自的文件系统，难以共享
    3) 分布式：系统中所有文件都可分布在不同处理机上，但逻辑上形成整体，用户无需了解具体物理位置即可存取。存取速度和文件保护有重要意义
4. 系统重构：自动切除故障资源并换上备份资源，若没有备份资源则重构系统是

指降级运行。若故障处理机上亟待运行的进程应安全转移至其他处理机

三. 多处理机操作系统的类型

1. 主从式master-slave
    1) 运行操作系统的特定处理机称为主处理机master processor
    2) 主处理机负责保持和记录所有处理机的属性、状态等信息，将从处理机视作可调度和分配的资源，为它们分配任务。从处理机无调度功能
    3) 工作流程：从处理机提交任务申请，等待主处理机发回应答；主处理机收到请求后中断当前任务，识别请求并转入对应处理程序，分配合适的任务给发出请求的从处理机。如CDC Cyber-170的外围处理机Po就是这种主处理机，又如DEC System10只有一台主处理机和一台从处理机
    4) 优缺点：
        (1) 易实现：只需适当扩充传统单机多道程序系统；除一些公用例程外，不需改写成可重入的，表格控制、冲突和封锁问题等都能简化
        (2) 资源利用率低：执行大量短任务时，请求任务队列较长，形成瓶颈
        (3) 安全性较差：主处理机发生不可恢复性错误易造成整个系统的崩溃
    5) 因此它只适合工作负载不重、处理机数量不多、只有一个处理机性能极高的非对称多处理机系统
2. 独立监督式separate supervisor system/独立管理程序系统
    1) 各处理机都有自己的管理程序/OS内核、IO设备和文件系统等专用资源、类似单机操作系统的管理和分配等功能的操作系统。如IBM370/158
    2) 优缺点：
        (1) 自主性强：各处理机可根据自身及任务的需要执行各种功能
        (2) 可靠性高：处理机相对独立，局部故障不会引起整个系统崩溃。不过缺乏统一管理和调度机制会使补救工作较困难
        (3) 实现复杂：管理程序的代码必须是可重入的，或者需要为每个处理机提供专用的管理程序副本；访问公用表格易发生冲突，需要仲裁
        (4) 存储空间开销大：每台处理机都有局部存储器驻留操作系统内核
        (5) 处理机负载不平衡：无主处理机负责管理和调度，难以平衡负载
3. 浮动监督式floating supervisor control mode/浮动管理程序控制方式
    1) 所有处理机组成一个处理机池，共享所有资源，某段时间内可以指定任一台（或更多台）处理机作为主处理机（组），根据需要，可以切换到其他处理机（组）。如IBM3081的MVS，C·mmp的Hydra等
    2) 优缺点：
        (1) 灵活：处理机可访问任何资源，大多数任务可在任一处理机运行
        (2) 可靠：任一处理机失效，只是少了一台可分配处理机而已，可切换
        (3) 负载均衡：可将IO中断等非专门操作分配给空闲处理机
        (4) 实现复杂：存储器模块和系统表格访问冲突需仲裁，如访问存储器用硬件解决，系统表格用优先级策略解决；多个主处理机同时执行的管理服务子程序需要有可重入性
            i.

ii.
iii.
iv.
v.
vi. ----------我是底线-------

10同步、调度

2019年2月14日 16:38

-
- 进程同步

1. 紧密耦合系统可直接通过共享存储时限同步

一. 集中式与分布式同步方式

1. 同步实体Synchronizing Entity：实现同步的实体，如硬件锁、信号量、进程
   1) 中心同步实体：满足以下条件的同步实体：
      (1) 有唯一的名字，被所有需同步的进程知道
      (2) 任何时刻都可被这些进程访问
   2) 中心同步实体的容错技术：失效时，系统会立即选一个新的投入运行
2. 集中式同步机构：基于中心同步机构的同步机构
   1) 对同一处理机的同步机制：硬件锁、信号量等
   2) 对不同处理机的：自旋锁、RCU锁、时间邮戳、事件计数、中心进程等
3. 集中式与分布式同步算法
   1) 集中式同步算法
      (1) 多进程访问共享资源或通信时，仅由中心控制结点判定出一个进程
      (2) 判定所需信息都集中在中心控制结点
      (3) 适用于单处理机系统和共享存储器的多处理机系统；分布式系统适合分布式同步算法；松散耦合式则两种都可能采用
      (4) 缺点：
         i. 可靠性差，结点故障易形成灾难性影响
         ii. 易形成瓶颈，管理大量资源的结点影响到系统的响应速度
   2) 分布式同步算法
      (1) 所有结点有相同信息
      (2) 所有结点仅基于本地信息做出判定
      (3) 为做出判定，所有结点担负相同职责，付出相同工作量
      (4) 一个结点发生故障通常不导致整个系统的崩溃
4. 中心进程方式
   1) 中心进程/协调进程：保存所有用户的存取权限、冲突图conflict graph等
   2) 访问共享资源前需要先向中心进程发送一条请求信息，中心进程查冲突图确认不会引起死锁后将该请求插入请求队列，否则退回rollback
   3) 当资源空闲，中心进程向队首进程发出消息，允许使用；释放资源时也要向中心进程发出消息。进出临界区必须有请求、回答、释放消息
   4) 为提高可靠性，中心进程可以浮动

二. 自旋锁spin lock

1. 自旋锁的引入
   1) 读-改-写操作需要执行多次总线操作，若执行原语过程中，总线被其他CPU争得，可能导致该存储单元被交叉操作，破坏原语的原子性

2) 自旋锁机制可用于但不局限于对总线资源的竞争

2. 实现互斥访问总线的方法
    1) 在总线设置一个自旋锁，该锁最多只能被一个内核进程使用
    2) 获得不到自旋锁的进程会“自旋”循环测试锁的状态，直至得到
3. 自旋锁与信号量的主要差别
    1) 自旋锁可避免调用进程阻塞，因为自旋锁保护的临界区一般较短，不会产生“忙等”，信号量机制切换进程花费开销大，使用自旋锁的效率更高
    2) 若不需中断上下文、或需共享设备、或临界区较大，还是该用信号量
    3) 自旋锁保持期间不可抢占，只在内核可抢占或SMP时才真正需要；单CPU可通过关中断防止中断处理的并发，自旋锁的操作是空操作
4. 自旋锁的类型
    1) 普通自旋锁：可获得时才为0，使用自旋锁不影响当前处理机的中断状态
    2) 读写自旋锁：允许被多个读者只读访问，需要读者数计数和解锁标记
    3) 大读者自旋锁：获取读锁时只需对本地读锁加锁，开销小；获取写锁时必须锁住所有CPU的读锁，代价高
    4) 基本形式：

```
spin_lock(&lock);
/*临界区*/
spin_unlock(&lock);
```

三. 读-拷贝-修改锁
1. Read-Copy-Update锁的引入
    1) 只要有一个进程在写，多个读进程会同时被阻塞，严重影响读工作
    2) 可以通过让写进程先用读（拷贝）出一个副本，对副本修改，再写回去
2. RCU锁
    1) 读不需任何锁，写只需制作出副本，再在适当时机利用回调机制
    2) 回调callback机制：向垃圾收集器机构注册一个回调函数，负责让原数据指向新数据，并释放数据
3. 写回时机
    1) 规定读任务结束后需要提供一个信号，所有读者都发送信号时即可修改
    2) 延迟期grace period：副本修改完到等待信号之间的时间
4. RCU锁的优缺点
    1) 读者不会被阻塞，提高了读进程的运行效率并减少了CPU上下文处理开销
    2) 无需为共享数据设置同步机构：对读者来说，没什么同步开销，也不怕死锁；不过写者需要复制文件、延迟释放、同步其他写操作
    3) 不适用于写操作多于读操作的情况，对读的性能提高可能不足以弥补写的损失，此时适合读写自旋锁

四. 二进制指数补偿算法和待锁CPU等待队列机构
1. 二进制指数补偿算法：对CPU测试锁的TSL指令设置延迟执行时间，该时间每次测试后扩大到该次延迟时间的两倍，直至到达一个最大值
    1) 可明显降低总线上的数据流量，因为减少了测试次数和频率

2) 缺点是延迟时间可能导致空闲的锁不能被及时使用，造成浪费

2. 待锁CPU等待队列机构：在每个CPU的高速缓存中配一个用于测试的私有锁变量和待锁CPU清单，访问共享数据失败的CPU会被分配到锁变量，并被添加到正在使用该数据的CPU的清单末，之后第n个CPU将被添加到第n-1个CPU后面

1) 当共享数据占用者退出临界区时，检查高速缓存，释放私有锁变量

2) 每个待锁CPU都仅在自己的高速缓存中不断测试私有锁，不访问总线

3) CPU释放锁后应及时释放清单下一个CPU的锁，避免浪费空闲资源

五. 定序机构

1) 多处理机系统和分布式系统中，每个系统都有自己的物理时钟

2) 定序机构负责排序各系统中所有特定事件，保证各处理机的进程协调运行

1. 时间邮戳定序机构Timestamp Ordering Mechanism：用系统中唯一的、单一物理时钟驱动的物理时钟体系，确保各处理机时钟严格同步。其基本功能：

1) 对资源请求、通信等特殊事件加印上时间邮戳

2) 对每种特殊事件只能用唯一的时间邮戳

3) 根据时间邮戳定义所有事件的全序

2. 事件计数Events Counts同步机构

1) 定序器sequencer：用于为所有特定事件排序的整型量

2) 定序器初值为0，非减少，只能被施加ticket操作

3) 事件刚发生时会被系统分配标号V，之后ticket自动+1

4) 系统会将已服务事件的标号保留，形成事件计数栈E，其值为栈顶标号

(1) await(E,V){ //进程进入临界区前需执行await

```
if(E<V){
    i=EP;
    stop();
    i->status="block";
    i->sdata=EQ;
    insert(EQ,i); //将执行进程插入EQ队列
    scheduler();
}else continue;}
```

(2) advance(E){ //进程退出临界区后需执行advance

```
++E;
if(EQ!=NIL){
    V=inspect(EQ,1);
    if(E==V)
        wakeup(EQ,1);}}
```

(3) read(E)：返回E当前值

(4) 以上三个操作在同一事件上可并发执行，但定序器必须互斥使用

六. 面包房算法

1) 是最早的分布式同步算法，通过给时间排序，再按FCFS处理

1. 系统由n个结点组成，每个结点仅有一个进程，仅负责处理一种临界资源

2. 每个进程保持一个队列，用来记录按事件时序排序的收到的消息和产生的消息
3. 消息分为：请求消息、应答消息、撤销消息
4. 进程Pi发送的请求消息形如request(Ti,i)。Ti=Ci是发送时逻辑时钟值，i是内容
5. 面包房算法描述：
    1) Pi请求资源时，把request(Ti,i)排在自己的请求队列中，并发给其他进程
    2) Pj收到消息后，放入自己的请求队列中，再发送回答reply(Tj,j)
    3) 满足以下条件时，允许Pi进入临界区/访问该资源
        (1) Pi的该请求消息处于请求队列最前
        (2) Pi收到所有其他进程发来的回答消息，时间戳均晚于Ti
    4) 释放资源时要从队列中撤销该请求，再发送打上时间戳的release消息给其他进程，收到该消息的进程也撤销队列中的该请求

七. 令牌环算法

1. 也是分布式同步算法，会将所有进程组成一个逻辑环Logical Ring
2. 令牌Token：特定格式的报文，在逻辑环中被循环传递，获得令牌的进程有权进入临界区访问共享资源，由于只能被一个进程持有，实现了互斥
3. 令牌初始化后随机赋予逻辑环任一进程，以点对点的形式按固定方向和顺序依次逐个传到下一个进程，不需访问共享资源的话就不保持令牌，直接传递
4. 缺点是令牌丢失或被破坏时难以检测和判断，如通信链路、进程故障等，需要及时屏蔽故障，重构逻辑环，重颁令牌等

-
- 多处理机系统的进程调度

一. 评价调度性能的若干因素

1. 任务流时间：完成任务所需时间
2. 调度流时间：系统中所有处理机的任务流时间综合
3. 平均流：调度流时间除以任务数
    1) 平均流小反应了资源利用率高，任务的机时费用低，完成任务时间充裕
    2) 最少平均流时间是系统吞吐率的间接度量参数
4. 处理机利用率：任务流之和除以最大有效时间单位
5. 加速比：各处理机忙时间之和除以并行工作时间（开始运行到全结束的时间）
    1) 加速比用于度量多处理机系统的加速程度
6. 吞吐率：单位时间内完成的任务数，一般用任务流最小完成时间来度量
    1) 与调度算法复杂性有密切关系
    2) 求解最优调度是Nondeterministic Polynomial完全性问题
    3) 难以求得最坏情况下的最优调度，一般只考虑典型情况的合适调度

二. 进程分配方式

1. 对称多处理机系统中的进程分配方式
    1) 静态分配Static Assignment方式：从开始执行到完成都在同一处理器上
        (1) 为每一处理器设置一专用的就绪队列，之后同单处理机系统
        (2) 调度开销小；各处理器忙闲不均
    2) 动态分配Dynamic Assignment方式：每次被调度都随机分到任一处理器

(1) 系统中仅设置一个公共就绪队列

(2) 消除了忙闲不均现象；适用于紧密耦合系统，但对松散耦合系统需要传递前一次运行时的处理器上下文，增加了调度开销

2. 非对称MPS中的进程分配方式：由主机为发出空闲信号的从机分配进程。为防止主机故障导致的系统瘫痪和主机太忙形成的系统瓶颈，可设置多台主机

三. 进程/线程调度方式

1. 自调度Self-Scheduling方式

1) 自调度机制：在系统中设置一个公共就绪队列，空闲处理机自行调度。由于多处理机使等待速度减小，FCFS的效率高于优先权调度

2) 优点：可以简单地沿用单处理机系统调度算法，不容易忙闲不均

3) 缺点：互斥访问队列易形成瓶颈；阻塞后的重运行需重新建立上下文，低效；合作性线程难以同时运行，导致切换频繁

2. 成组调度Gang Scheduling方式

(1) 由Leutenegger提出，将同一进程的线程成组分配

(2) 合作线程的并行执行能减少阻塞，改善系统性能

(3) 每次调度可解决一组线程，减少调度频率和开销，性能优于自调度

1) 面向所有应用程序平均分配处理器时间：M个进程各被分配约1/M的时间

2) 面向所有线程平均分配处理机时间：N个线程各被分配约1/N的时间

3. 专用处理机分配Dedicated Processor Assignment方式

1) 1989由Tucker提出，在运行期间，专门分配一组处理机，直至完成

2) 易造成处理机空闲的严重资源浪费，不适用于处理机少的环境

3) 但对数十个处理机高度并行的系统来说，各处理机的投资费用只占很小一部分，每个线程专用一台处理机可有效避免切换，加速运行

(1) 如16个处理机的系统运行矩阵相乘和傅立叶变换（FFT）两个程序

(2)

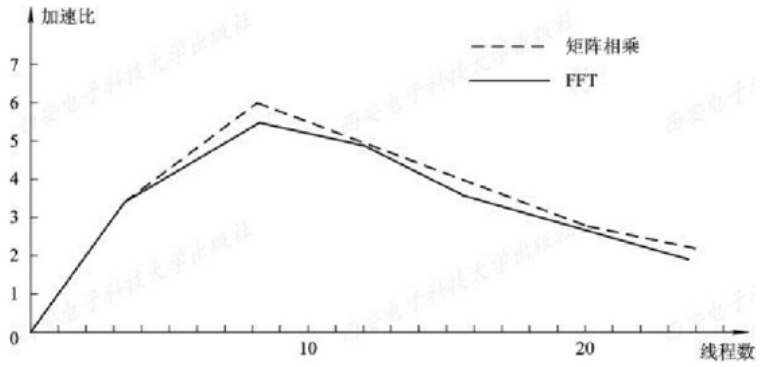


图10-8 线程数对加速比的影响

(3) 如图，线程数总和接近处理机数时加速比最高，不能保证线程数<=处理机数时，线程数越多，加速比越低。类似物理块数少于工作集数时会频繁导致缺页

4. 动态调度

1) 允许进程在执行期间动态改变线程数，由系统和程序共同进行调度决策

2) 系统负责分配处理机给作业，作业自行分配处理机执行部分任务

3) 系统分配处理机的原则

(1) 空闲则分配：有作业提出处理机请求时，将空闲处理机分配给作业
(2) 新作业绝对优先：优先分配给无任何处理机的新到作业，必要时收回分配给旧作业的处理机
(3) 保持等待：指系统任何分配都不能满足作业时需保持等待新处理机（或由作业自己取消处理及请求）
(4) 释放即分配：释放处理机后立刻分配给新作业，再按FCFS

4) 优于前两种调度，但开销过大

四. 死锁

1. 死锁的类型
   1) 资源死锁：竞争可重用资源（打印机、存储器等）或推进顺序不当。如集中式系统中互相等对方发送消息
   2) 通信死锁：主要是分布式系统不同结点中的进程因报文而竞争缓冲区
2. 死锁的检测和解除
   1) 集中式检测：各处理机都有一进程资源图描述进程及其占有资源，再由中心处理机配置全系统的进程资源图，并设置检测进程及时中止环路的进程
      (1) 检测方式：
         i. 当资源图中加入或删除弧时将相应变动消息发给检测进程
         ii. 由进程自行将弧的变动信息周期性地发送给检测进程
         iii. 检测进程主动请求更新信息
      (2) 缺点是发出消息与执行命令时序不一定相同。可以在检测到环路后重新向进程发出请求，若收到了否认的回答就说明未死锁
   2) 分布式检测：系统中竞争资源的进程相互协作，自行检测
      (1) 在每个结点中都设置一个检测进程，每个消息上都附加逻辑时钟，并依次对请求和释放资源的消息排队
      (2) 请求某资源前需要给所有其他进程发送请求信息，获得全部响应后才能把请求资源的消息发给管理进程，分配情况也要通知所有进程
      (3) 可见，分布式环境死锁检测的通信开销较大，一般是靠死锁预防
         i.
         ii.
         iii.
         iv.
         v.
         vi.
         vii. ---------我是底线-------

10网络

2019年2月17日 16:38

-
- 网络操作系统

1. 自主计算机：有独立处理能力的计算机
2. 计算机网络：通过数据通信系统将地理上分散的自主计算机系统连接起来，达到数据通信和资源共享目的的计算机系统

一. 网络及网络体系结构

1. 计算机网络的组成
   1) 物理结构：通过星形、树形、总线形、环形、网状形等拓扑结构将地理上分散的计算机连接起来的网络
   2) 逻辑结构：
      (1) 通信子网：由分布在不同地点的、负责数据通信处理的通信控制处理机与通信线路互连构成。是基础部分，负责数据传输及交换
      (2) 资源子网：由主计算机与终端构成，每个信源和信宿都连接在其中的交换设备上。主计算机负责数据处理
      (3) 网络协议：为实现数据交换而建立的规则、标准或约定的集合，保证网络中源主机系统和目标主机系统保持高度一致的协同
2. 网络协议
   1) 是一组控制数据交互过程的通信规则，规定了通信双方交换数据/控制信息的格式和时序，其三要素为：
      (1) 语义：解释控制信息各部分的意义，规定双方要发出的控制信息、执行的动作和返回的应答等
      (2) 语法：即确定协议元素的格式，包括用户数据和控制信息的结构与格式及数据的顺序，规定了双方如何操作
      (3) 时序：对时间发生顺序的详细说明，指出事件的顺序和速率匹配等
   2) 网络每层都有相应的协议，决定了该层实体在执行功能时的行为
   3) 内聚性高，耦合性低：协议应处理其他协议没处理过的通信问题，为此，各协议间可以共享数据和信息
3. 互联网协议IPv4和IPv6
   1) IPv4协议
      (1) 寻址：为每个实体赋予唯一的全局性标识符
      (2) 分段和重新组装：信息在不同帧长度的网络中交换前后的工作，如X.25网信息最大长度为128字节，以太网为1518字节
      (3) 源路由选择：为IP数据报的传输选择最佳的传输路由
   2) IPv6协议：保留IPv4优点的同时针对不足做出修改，更适合当今互联网的需要，如地址空间从4字节扩充到16，引入了认证技术等安全机制
4. 传输层协议
   1) 传输控制协议TCP

(1) 提供了面向连接的、可靠的端-端通信机制
(2) 可靠：即使网络层（通信子网）出了差错，仍能正确连接、传输、释放，且发送速度和接受速度有流量控制

2) 用户数据包协议UDP
(1) 是无连接的、不可靠的协议，无需建立或拆除端-端连接
(2) 传输过程不对数据进行差错检测，较简单，传输速率高

5. 网络体系结构
(1) 层次结构：把复杂的系统设计问题分解成多个层次分明的局部问题，并规定每层必须完成的功能
(2) 网络体系结构是抽象的，实现网络协议的技术是具体的
(3) 开放系统互连参考模型Open System Interconnection/Reference Model参考模型层次划分原则：
i. 网络中各主机都有相同层次
ii. 不同主机的同等层有相同功能
iii. 统同一主机的相邻层间通过接口通信
iv. 每层可以使用下层提供的服务，并向上层提供服务
v. 不同主机的同等层通过协议来实现同等层之间的通信
(4) OSI体系结构从低到高分别是：

1) 物理层Physical Layer：建立在通信介质的基础上，实现系统和通信介质的接口功能，为数据链路实体之间透明地传输比特流提供服务
2) 数据链路层Data Link Layer：在相邻两系统的网络实体之间，建立、维持和释放数据链路连接，实现透明、可靠的信息传输服务。传输单位是帧
3) 网络层Network Layer：主要有通信子网及与主机的接口，提供建立、维持和释放网络连接的手段，实现传输实体间的通信。传输单位是组packet
4) 传输层Transport Layer：为不同系统内的会晤实体间建立端-端end-to-end透明、可靠的数据传输，执行端-端差错控制和流量控制，管理多路复用等。传输单位是报文message
5) 会晤层Session Layer：为不同系统的应用进程间建立会晤连接。增值了基本的传输连接服务，提供了能满足多方面要求的会晤连接服务
6) 表示层Presentation Layer：向应用进程提供信息表示方式、对不同系统的表示方法进行转换，使不同表示方式的应用实体间能进行通信，并提供标准的应用接口和公用通信服务，如数据加密、正文压缩等
7) 应用层Application Layer：为应用进程访问OSI环境提供了手段，其他层通过该层直接为应用进程服务

8)

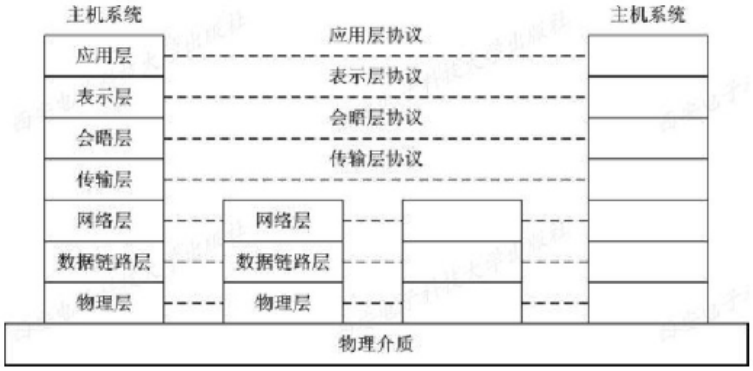


图10-9 OSI七层模型

二. 网络操作系统及其分类

1. 网络操作系统及其特征

1) 网络操作系统Network Operating System：在计算机网络环境下，对网络资源进行管理和控制，实现数据通信及对网络资源的共享，为用户提供与网络资源之间接口的一组软件和规程的集合

2) 建立在网络中计算机各自不同的单机操作系统上，提供使用网络系统资源的桥梁，主要特征有：

(1) 硬件独立性：系统可运行于各种硬件平台上

(2) 接口一致性：对同一性质的网络中的共享资源采用统一的访问方式和接口

(3) 资源透明性：对网络中的资源统一管理，能根据用户要求自动分配

(4) 系统可靠性：通过统一管理、分配和调度手段，确保了整个网络的安全可靠，屏蔽故障节点或重新定义新的通信链路即可正常运行

(5) 执行并行性：各节点中各进程并发，网络中各节点并行

2. 网络操作系统的分类

(1) 主从模式、对等模式、基于服务器模式

1) 对等模式peer-to-peer model：各节点地位平等，无从属关系

(1) 任两节点都能直接通信，任一计算机都能访问共享资源

(2) 任一计算机都分前台本地服务和后台为其他节点服务两种工作

(3) 适用于小范围网络，结构简单，安装维护简单

(4) 计算机负载过重，既要承担本地用户的信息处理任务，又要承担较重的网络共享资源管理和通信任务，降低了信息处理能力

2) 工作站/服务器模式Workstation/Server model：将节点分为两类：

(1) 工作站：为本地用户提供访问本地资源和访问网络资源服务

(2) 服务器：以集中方式管理网络中的共享资源，为工作站提供服务

(3) 服务器是局域网的逻辑中心，安装其上的OS软件的功能和性能决定了网络服务功能的强弱及系统的性能和安全性，是网络操作系统的核心部分，又称基于专用服务器的网络操作系统

(4) 多数网络都采用了非对等结构的网络操作系统，用于大型系统的联网，如Novell的NetWare和微软的WindowsNT

3) 客户/服务器模式Client/Server model：将任务分配在两台或多台机器上

(1) 客户机提供用户接口和前端处理，并向服务器提出资源、数据和服务请求
(2) 服务器接受请求并提供资源、数据和服务
(3) 软件上，应用或软件系统也需按逻辑功能划分
(4) 客户端软件负责数据表示和应用、用户界面的处理、接受用户的数据处理请求、向服务器发出数据和服务请求
(5) 服务器软件负责接受请求并执行相应服务
(6) 客户机和服务器间采用TCP/IP、IPX/SPX等协议连接和通信
(7) 交互模式：发请求，接请求，发响应包，接响应包

4) 浏览器/服务器模式Browser/Server model
(1) 配置了浏览器软件的客户机可以浏览网络中几乎所有允许访问的服务器，这种客户机又称Web浏览器
(2) 传统的小型局域网可以采用两层C/S模式，大型网络中通常采用三层
(3) 三层C/S模式中客户机访问的是Web服务器，由其代理访问服务器
(4) 浏览器与服务器的交互方式也是请求/响应方式，用于浏览器检索的对象通常是超文本文件，所以采用http协议

三. 网络操作系统的功能

1. 数据通信
1) 建立与拆除连接（物理层、数据链路层、网络层的连接）
2) 报文的分解与组装：在网络层和传输层之间传送时的转换
3) 传输控制：为报文配上报头信息，如目标地址、源主机地址、报文序号
4) 流量控制：控制发送速度不超过目标接收和处理能力、控制链路传输能力
5) 差错的检测与纠正（指传输过程中的差错）

2. 应用互操作
1) 信息的互通性：为避免协议不同导致无法识别和通信，应配置同一类的传输协议，如Internet由各类WAN互联而成，通过TCP/IP协议实现互通
2) 信息的互用性：网络中的用户可访问其他网络文件系统中的文件或数据库系统中的数据，如同一使用SUN的Network File System协议（不能访问是因为文件系统的结构、文件命名方式、存取命令等不同）

3. 网络管理
1) 网络管理的目标
(1) 增强网络可用性：预测网络故障，配置冗余设备等
(2) 提高网络运行质量：随时监测网络中的负荷及流量等
(3) 提高网络资源利用率：长期监测网络、合理调整网络资源等
(4) 保障网络数据安全性：采取多级安全保障机制等
(5) 提高网络的社会和经济效应等

2) 网络管理的功能
(1) 配置管理：监控网络的配置数据，允许网络管理人员能生成、查询和修改软硬件的运行参数和条件，以保证网络正常运行
(2) 故障管理：检测网络异常，提供为了操作员发现和修复手段

(3) 性能管理：收集网络运行数据、分析运行情况及运行趋势、得出对网络的整体和长期的评价，将性能控制在用户能接受的水平

(4) 安全管理：提供安全策略来控制对资源的访问，包括定义合法操作员及其权限和管理域，规定用户对网络资源的访问权限、通过使用日志等手段调查关系的事件、防止病毒等

(5) 计费管理：监视和记录用户使用网络资源的种类、数量和时间、对资源的使用计费

i.

ii.

iii.

iv.

v.

vi.

vii.

viii.

ix.

x. ---------我是底线-------

10分布

2019年2月19日 17:15

-
- 分布式文件系统

1. 本地文件系统Local File System：仅允许本地CPU通过IO总线访问存储设备
2. 分布式文件系统Distributed File System：建立在松散耦合MPS上的，通过通信网络或计算机网络互连，将分布在若干节点的存储设备以共享文件系统方式统一管理，提供给不同节点上的用户共享
   1) 基于客户机/服务器模式，是相对独立的软件系统，被集成在分布式系统

一. 分布式系统distributed system

1) 定义：是基于软件实现的一种MPS，多处理机通过通信线路互连而构成松散耦合系统，系统的处理和控制功能分布在各处理机上
2) 或者说，是利用软件系统构建在计算机网络上的一种MPS
3) 又或者，分布式系统是独立计算机的集合，用户体验却像一台计算机
4) 又或者，是能为用户自动管理资源的网络操作系统，负责调用完成用户任务所需资源，整个网络像一个大的计算机系统一样，对用户透明
5) 与前述的MPS区别有
   (1) 各节点都是独立计算机，配有完整的外部设备
   (2) 节点耦合程度更分散，地理分布区域更广阔
   (3) 各节点都能运行不同的操作系统，有各自的文件系统，除了本节点的管理外，还有其他多个机构对其实施管理

1. 分布式系统的特征
   1) 分布性
      (1) 计算机地理位置分布性：位置和地域范围分散且广阔
      (2) 系统功能的分布性：分散在各节点计算机上
      (3) 系统资源分布性/自治性：分散配置在各节点计算机上
      (4) 系统控制的分布性/自治性：各计算机一般无主从之分
   2) 透明性：CPU、文件、打印机等，被共享，用户可使用其他节点的资源
   3) 同一性：程序可分布在多台计算机上并行运行，即计算机可协作完成任务
   4) 全局性：系统具备全局进程通信机制，任两台计算机可交换信息
2. 分布式系统的优点
   (1) 缺点是专用软件少，网络安全的潜在不足多
   1) 计算能力强：并行处理技术，使任务可被划分、分配给多节点并行执行
   2) 易实现共享：基于计算机网络使资源易共享，计算迁移功能使负载易共享
   3) 方便通信：基于计算机网络和地域范围广阔为信息交流提供了方便
   4) 可靠性高：工作负载分数使单个机器故障不影响其他机器
   5) 可扩充性好：不需修改系统软硬件即可添加若干台计算机
3. 分布式操作系统
   1) 是配置在分布式系统上的公用操作系统，以全局的方式对分布式系统中的

所有资源进行统一管理，直接统一管理系统中地理位置分散的各物理和逻辑资源，有效协调和控制各任务的并行执行，协调和保持系统内的信息传输及协作运行，向用户提供统一的、方便的、透明的使用系统的界面和标准接口。典型的例子是world wide web及其唯一操作页面：web页面

2) 与网络操作系统不同，分布式系统用户使用系统内资源时不需了解各计算机的功能与配置、操作系统的差异、软件资源、网络文件的结构、物理设备的地址、远程访问的方式等，系统内部实现的细节也对用户屏蔽
3) 保持了网络操作系统功能的同时具有透明、内聚、可靠、高性能等特点
4) 除单机系统具有的功能以外还应实现
   (1) 通信管理：提供通信机制和方法，使不同结点的进程能方便地交换信息。一般是通过提供符合通信协议的原语实现通信
   (2) 资源管理：统一管理、分配、调度所有系统资源，方便用户使用，提高资源利用率。如分布式文件系统，分布式数据库系统、分布式程序设计语言及编译系统、分布式邮件系统等
   (3) 进程管理：提供进程或计算迁移，实现平衡结点负载，加速计算速度；提供分布式同步和互斥机制，实现协调资源共享和竞争，提供进程并行程度，应对死锁

二. 分布式文件系统的实现方式和基本要求

1. DFS的实现方式
   1) 共享文件系统方式shared file system approach/专用服务器方式
      (1) 类似本地文件系统的树形目录结构，用逻辑树结构管理整个系统
      (2) 客户/服务器模式，设置若干的文件服务器，数据分布其中
      (3) 用户无需关注物理位置，只需按逻辑关系，即可通过文件服务器的服务进程，访问整个系统的共享资源
      (4) 服务器分布和逻辑树架构对用户透明。如NFS、AFS、Sprite等系统
      (5) 实现简单，设备要求低，通用性好
   2) 共享磁盘方式shared disk approach/无服务器方式
      (1) 配置一共享磁盘，与主机、客户机都连接在内部高速网络（如光通道上），主机和客户机都将共享磁盘视作其存储设备，以盘块方式读写磁盘上的文件，实现共享，如VAX Cluster、IBM GPFS和GFS
      (2) 设备要求高，实现难度大，一般用来构造高端货专用存储设备如Network Attached Storage和Storage Area Network
2. 基本要求
      (1) 相比LFS，DFS除了大容量外，还要求
   1) 透明性：以下特点对用户透明
      (1) 位置：文件服务器及文件服务进程的多重性，共享存储器的分散性
      (2) 移动：存储资源和数据在系统内位置的移动
      (3) 性能：系统中的服务负载有一定范围的变化量，保障性能稳定
      (4) 扩展：在规模、性能和功能上允许系统扩展以满足负载和网络规模
   2) 高性能和高可靠性，满足诸多客户繁重的访问需求，保证系统安全可靠

3) 容错性：保证信息可靠性，系统出错时仍能为用户提供服务，一般通过冗余技术实现故障的掩盖：信息冗余，如增加校验信息；时间冗余，如重复执行操作；物理冗余，如增加额外的副本、设备或进程等
4) 安全性，通过身份验证、访问控制和安全通道等
5) 一致性，保证客户在本地缓存的文件副本与文件服务器上的一致

三. 命名及共享语义

1. 命名：除了LFS的屏蔽盘面、磁道号、扇区的抽象外，还需隐藏服务器地址和存储方式等细节，提供多级映射的抽象访问名接口
   1) 结合主机名和本地名，简单易行但不透明，如/host/local-name-path
   2) 将若干服务器的远程目录加载到客户机的本地目录。复杂度高，结构混乱，安全度低，一个服务器故障会导致客户机上的目录集失效
   3) 全局统一命名，每个文件和目录使用唯一名。考虑不到特殊文件，难实现
2. 共享语义
   (1) 为保证多客户机并发访问时的数据一致性，必须精确处理客户机和服务器之间的交互协议，即精确定义读和写
   1) UNIX语义：文件的每个操作对所有进程都是瞬间可见
   2) 会话语义：文件关闭前，所有改动对进程都不可见
   3) 不允许更新文件，只能简单共享和复制
   4) 事务处理：所有改动以原子操作的方式顺序发生
3. 租赁协议
   1) 是一种有代表性的一致性访问协议，是一种多读者单写者机制，收到待读数据的同时会收到一个租赁凭据，上面附带保证数据不会被更新的有效期
   2) 在该有效期内读数据可直接访问在本地缓存的副本，过期后需要向服务器确认是否发生了更新，是否要接受新数据
   3) 服务器收到了更新请求时，需向所有该数据的租赁客户机发出更新确认，收到更新确认的客户机立刻标记凭据过期

四. 远程文件访问和缓存

1) 根据程序的局部性原理，DFS调用远程过程调用RPC，送回给本地缓存的数据量会比实际请求的多得多

1. 缓存和远程服务的比较
   1) 将远程访问转为本地缓存访问，加快了访问速度
   2) 使用本地缓存，减少了服务器负载和网络通信量，加强了扩充能力
   3) 使用本地缓存，一般减少了网络总开销
   4) 使用缓存带来了一致性问题，频繁写的情况下使上述方面优势转为劣势
   5) 使用本地缓存使仿真集中式系统的共享语义难定义；使用远程服务时，所有访问被系统串行化，能实现任何集中的共享语义
   6) 机器间的接口不同：远程服务方式仅是本地文件系统接口在网络上扩展，季期间的接口是本地客户机和文件系统间的接口；而缓存方式，数据实在服务器和客户机间整体传输，机间接口与用户接口不同
2. 缓存的粒度（数据单元）：可以是文件的若干块，可以是若干文件

1) 数据粒度越大，存储的数据就越多，就越增快了访问效率，减少了通信流量和服务器负载；但传输的一次性开销也会增大，不适用与频繁写的系统
2) 数据粒度小则降低了换成的命中率，增加了通信开销
3) 同理，存储块大小划分也会影响存储性能

3. 缓存的位置
   (1) 客户机和服务器的磁盘、主存，共四个位置
   1) 磁盘：可靠，不因系统故障而丢失，提高单个机器的自治性
   2) 主存：支持无盘工作站、访问速度快、方便构造单缓存机制（客户机和服务器都用主存换成时，就可使用单缓存，协议简单，易实现）
4. 缓存的更新
   1) 直接写：修改完立刻写回服务器磁盘，可靠性高
   2) 延迟写：先修改到缓存，等段时间再写回，可能（共享）语义不清
   3) 驱逐时写：修改过的数据被从缓存中换出时写回
   4) 周期性写：周期性扫描缓存，把上次扫描后修改的块写回
   5) 关闭时写：关闭文件时写回，对应会话语义
5. 数据一致性
   1) 客户机发起检查，与服务器联系
      (1) 检查频度可以是每次访问前进行、打开时进行、周期性进行
      (2) 检查频率直接影响了网络和服务器的负载
   2) 服务器发起检查
      (1) 需要记录每个客户机缓存的文件
      (2) 违背了客户/服务器工作模式

五. 容错

1. 无状态服务和有状态服务
   1) 有状态服务stateful file service：服务器向客户机提供服务时，缓存该客户机的有关信息，该服务器为有状态服务器
      (1) 服务过程中的崩溃使所有易失性状态丢失，难以恢复
   2) 无状态服务stateless file service：不缓存客户机信息，称无状态服务器
      (1) 系统崩溃后仍能很快重新向客户机提供服务
      (2) 但不保留客户机信息使请求消息变长，处理过程变久
2. 容错性
   1) 可恢复性：某文件操作失败或客户端中断操作时，可转换回原来的一致性
   2) 坚定性：某存储器崩溃或存储介质损坏时任能保证文件完好
   3) 可用性：无论何时都能访问，包括及其崩溃和通信失效时
      (1) 坚定性可用原子性操作保证，坚定性可用设备冗余保证
3. 可用性与文件复制
   1) 通过对文件在多服务器上独立备份，增强了文件的可用性
   2) 还可将文件访问请求分流到各服务器上，平衡负载，避免性能瓶颈
   3) 文件复制需要对用户透明，只在底层用标识符区分
   4) 文件更新协议复杂，为保证一致性会付出大量开销

i.
ii.
iii.
iv.
v.
vi.
vii.
viii.
ix.
x.
xi.
xii. ----------我是底线-------